SCHEDULING AND ASSIGNMENT ON DYNAMIC PROCESSOR NETWORKS

A Thesis
Presented to
The Academic Faculty

By

John B. Mains

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in
Electrical and Computer Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology

August 2020

SCHEDULING AND ASSIGNMENT ON DYNAMIC PROCESSOR NETWORKS

Approved by:

Dr. Eric Feron, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Samuel Coogan
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Matthieu Bloch
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: July 24, 2020

ACKNOWLEDGEMENTS

I would like to thank: Dr. Feron for guiding me towards what has been ultimately fulfilling work, Dr. Coogan and Dr. Bloch for helping me to see what this work should really be, my labmates for their assistance and good company, General Atomics for the gift that allowed me to conduct this work, and finally my family for their continued support during my studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

We address the problem of scheduling tasks characterized by dynamic completion time behavior. Existing scheduling methodologies reduce problems to using constant task durations and are therefore not suitable for problems with more complex completion time behavior. One such problem is scheduling applications on computer networks with time-varying communication channel capacities.

We present a methodology that can be used to solve such problems. The specific contributions of this work are: (a) a novel representation for the completion time of tasks in scheduling problems, (b) a method for approximating that representation efficiently using an affine over-approximation, and (c) a mixed-integer programming-based formulation to solve a variety of scheduling problems using this representation and approximation. The resulting framework can be used to solve scheduling problems with time-varying task durations, and thus solve problems with time-varying communication links or resources.

A case study is provided, in which we devise a scheduling scheme for computation on a computer network composed of processors onboard a satellite constellation. This example is an instance of scheduling applications on computer networks with known time-varying communication channel capacities and the results of the case study demonstrate the improvement of using the approximation method over existing methods.

CHAPTER 1
INTRODUCTION

1.1 Motivation

In recent years, onboard processing and networking capabilities available for vehicles have become sufficient to enable the processing of complex real-time mission-critical applications using an offboard processor, as well as the processing of applications originating from an offboard source onboard the vehicle [1][2]. The concept of processing mission-critical applications offboard is referred to as offloading. In the context of robotics offloading is studied as cloud robotics [3], of which there exist several practical implementations [4][5]. Most of these, however, focus on offloading within the existing cloud paradigm, in which the offboard resources are centralized in a datacenter. However, if the offboard processors and sources are other vehicles within the same networked group, offloading within the group is attractive for vehicles whose regular operation varies with circumstance, such as allowing load balancing within the network, or faster response times for mission-critical applications shared amongst the group. Also, if a vehicle within the group suffers a failure in either networking or processing hardware, the vehicle's mission-critical applications can be offloaded within the network to ensure continued operation, as long as network connectivity is maintained.

However, to efficiently offload within the network, the scheduler needs to account for changes in network parameters, such as available processor and networking resources. In order to account for changes in network parameters, the changes must be known prior to scheduling. In certain problems, the evolution of the network parameters can be predicted with certainty, such as in the case of a satellite constellation. For other cases, such as in a platoon of cars, network state estimation is a problem that must be first solved. Estimating

these changes for vehicular networks has been solved by predicting the evolution of the network based on the environment knowledge and the paths of the component vehicles [6]. There has been work on parameter estimation and prediction for networks based on groups of unmanned aerial vehicles [7]. Predicting the behavior of general mobile networks has shown promising results, expanding the potential application of distributed computing on mobile processors to agents whose paths are uncontrolled [8].

There has been recent interest in finding scheduling strategies for offloading problems involving vehicles. One approach has focused on path planning with offloading objective formulated around static resources such as cell towers, [9][10]. Another has solved problem-specific communication-aware task planning, which is promising, but narrow in scope [11][12][13][14].

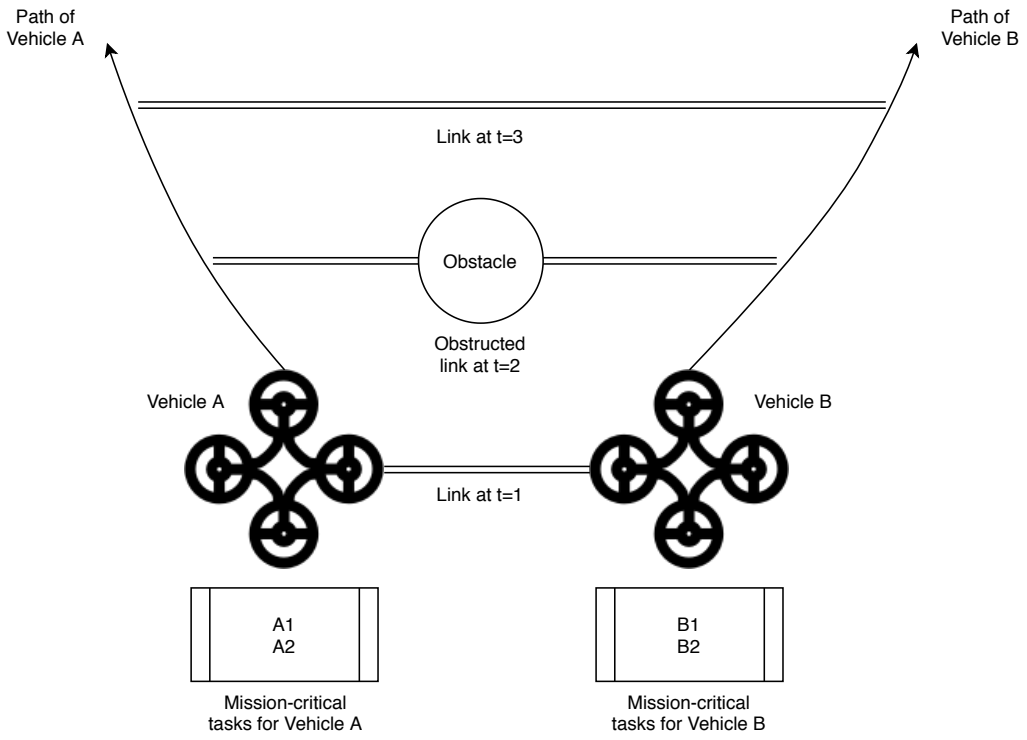


Figure 1.1: Visualization of offloading within a network.

We instead approach the scheduling and assignment of processing tasks within these

vehicle networks from the perspective of adapting scheduling theory to accommodate time-varying completion time behavior and formulating a mixed-integer program (MIP) to solve for the schedule.

1.2 Problem setting

To solve the problem of efficiently managing onboard resources and applications, we focus on scheduling and assignment problems for time-varying networks, with a particular focus on solving problems involving message passing across channels with time-varying capacities.

In this context, a *scheduling problem* consists of finding a mapping between a set of tasks and a positive interval, which can be thought of as time. The mapping of the output may represent the starting time of a task, the completion time, or the interval of operation. In this work, we look for mappings that yield the starting times of each task. The result of solving a scheduling problem is a *schedule*.

A major distinction in scheduling schemes is whether they are static or dynamic. In a static scheduling scheme, all tasks and their parameters are assumed to be known prior to the computation of the schedule and subsequent operations, and the scheduling of all tasks happens prior to execution of any tasks. In a dynamic scheduling scheme, tasks are assumed to arrive sporadically, and the scheduling of a task occurs at arrival. In this thesis, the focus is on static scheduling schemes. An assignment problem consists of finding a mapping between a set of tasks and a set of targets. In scheduling problems with multiple machines, otherwise known as scheduling and assignment problems, the set of targets represents the machines or workers available, and the result of solving such problems is a schedule and an assignment of every task to a machine. In the context of scheduling and assignment, the resulting assignment mapping need not be onto or one-to-one.

A schedule is feasible if every task has a corresponding starting time in the resulting schedule and all of the constraints of the problem are met. Likewise, if there are mul-

tiple machines, every task must have a machine assignment. Constraints of the problem may include additional concepts such as task-specific deadlines, fairness requirements, and restrictions on machine assignment. A particular scheduling problem is feasible if there exists a feasible schedule for that problem.

The constraint that is present in every scheduling problem is the no-overlap constraint, which requires that the runtime intervals of two tasks on the same machine cannot intersect. Thus any schedule that meets the no-overlap constraint will be onto time. The runtime interval of a task represents the time that a task occupies a machine. In problems without preemption, the runtime interval is defined by the starting time and completion time of a task. Preemption allows a task to override another task during execution, so the scheduled time for tasks is no longer guaranteed to be contiguous. In this thesis, we do not consider problems with preemption.

The no-overlap constraint combined with a shared deadline for all tasks yields the base problem class. Problems within the base problem class are defined by the number of tasks N, the scheduling window length W, and completion time behavior that is the same for every task and is well-defined over the scheduling window. A feasible schedule to a problem within the base problem class satisfies the no-overlap constraint, and has a starting time and completion time for every task that lies within the scheduling window, defined as $D = [0, W]$. The completion time of a task is determined by its starting time and completion time behavior.

Graham's taxonomy is a formal classification of scheduling problems, which we use as a reference to establish problem variations [15]. Common to all scheduling problems within Graham's taxonomy is a set of N jobs to be scheduled on a set of M machines using some optimality criterion. Hereafter, we use the terms 'job' and 'task' interchangeably and assume the same meaning. The optimality criteria of scheduling problems within the taxonomy are functions of the completion time C_i of each task i. Within Graham's taxonomy, a problem can be defined as having unit processing time, in which each job has identical

processing time. Within this thesis, we refer to this type as a problem having uniform tasks, and nonuniform tasks if this is not the case. If certain jobs require that other jobs must complete before initiation, then the problem has precedence between jobs, otherwise known as the problem having task precedence or precedence relations. When the processing time for a given task is the same across every assignable machine, the problem is referred to as having identical parallel machines. Within this thesis, we refer to this as problems having uniform machines, or nonuniform machines if this is not the case. In problems where a given job can only be assigned to a subset of the machines, the problem has heterogeneous machines, otherwise homogeneous. In situations where the subset of machines is partly determined by the assignments of preceding jobs, the problem has machine precedence.

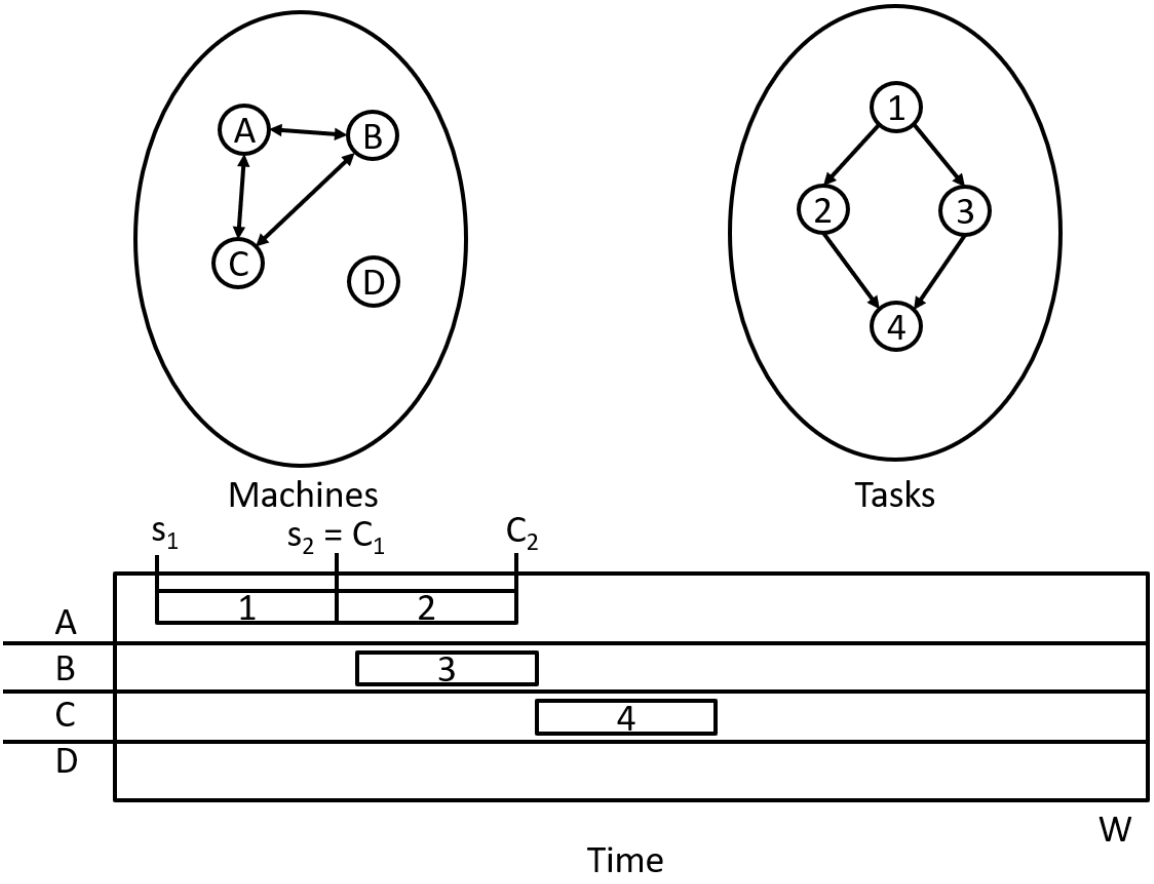


Figure 1.2: Visualization of a scheduling problem and a feasible schedule with uniform tasks, precedence relations, multiple uniform machines, and machine precedence.

In this thesis, we examine problems that use makespan as the optimality criterion.

Makespan is the duration from the start of the scheduling window to completion of the latest task to complete. For problems with makespan as the optimality criterion, if the makespan of a schedule is greater than the length of the scheduling window, the schedule is infeasible.

Scheduling problems with more than two machines are known to be NP-complete [16], and scheduling and assignment within a time-varying network are at least NP-complete.

The scheduling problems described in Graham's taxonomy are generally applicable to a range of topics, including operations and communications, as are the results we present. However, the motivating problem of this thesis pertains to scheduling computer processors, so we reference work focused on solving scheduling problems for computer processor networks. Scheduling with multiple machines in the context of computer processors is well studied and known as multiprocessor scheduling. Initial work by Shen [17] and Sinclair [18] focused on static assignment in distributed computer systems. Static assignment can be thought of as a scheduling and assignment problem where the tasks never complete. True scheduling and assignment on processor networks was first considered by El-Rewini [19]. Kwok examined the same problems with further modifications on the task and machine requirements, including precedence relations [20].

An important consideration in scheduling on processor networks is non-negligible communications, whether considered as delay, cost, or separate tasks. Mixed-integer programming based scheduling incorporating communication costs or delay has been recently investigated by Davidoc [21] and Vengopalan [22]. Another approach to handling communication is to schedule message passing tasks along with the processor tasks. Metzner developed a SAT formulation for scheduling on distributed real-time systems considering task precedence constraints, memory, and a time-division multiple access CAN-bus style communication paradigm, a form of message passing scheduling [23]. Of interest to our motivations is work on hard real-time scheduling in multiprocessors systems [24], and on hard real-time scheduling with communication constraints by Peng [25]. Abdelzahar solved for

a variant of this approach for distributed real-time systems involving real-time channels and synchronization constraints[26]. Craciunas developed a SAT and mixed-integer program based static scheduling scheme for problems involving communication over switched multi-speed networks [27].

The focus of this work is on scheduling problems with time-varying parameters. That is not to say the challenges in dynamic systems have not been considered. A mixed-integer linear programming formulation for allocations of tasks with uncertain execution times on a static network has been solved [28], as has scheduling and assignment on networks with uncertain time-varying nodes [29]. Time-varying network parameters are not foreign to the field of network channel scheduling either, dynamic power allocation and routing has been solved for time-varying wireless networks [30].

However, none of the work currently addressing scheduling for time-varying systems considers problems where the time-varying nature of the completion time behavior is well-defined, which is what we consider in this thesis.

1.3 Overview of thesis

The contributions of this thesis are a novel completion time model for scheduling problems, a method to approximate problems within the model to yield efficient scheduling solutions, and mixed-integer programs to solve a variety of scheduling problems using the approximation method, demonstrated by a case study.

The main body of this thesis is organized into three chapters. Chapter 2 consists of our novel completion time model, approximation methods, a mixed-integer program based scheduling scheme, and results relating to scheduling using the approximations. Chapter 3 consists of an introduction to scheduling problem variations, the associated mixed-integer program modifications required to schedule them, and numerical results demonstrating the benefits of the approximation method for problems with these additional constraints. Chapter 4 consists of an examination of a case study, along with the solution for a particular

instance and a comparison of the resulting schedules.

CHAPTER 2

COMPLETION TIME MODEL

In scheduling problems, the completion time of a task is determined by its starting time, assignment, and completion time behavior; that is, the time at which a task is completed is dependent upon the time at which the task starts, the machine that the task is assigned to, and the mapping from the starting time and assignment to the completion time. Existing methods for scheduling assume the completion time behavior of a task i is defined by the task's processing duration d_i. Thus, the completion time of a task is constrained to the form $C_i = s_i + d_i$, where s_i is the starting time of task i. Additionally, d_i is typically assumed to be the duration of task i in the worst case. When the processing time of a tasks varies in time itself, e.g. dynamic voltage and frequency scaling processing or physically determined wireless links, a constant worst case processing time based scheduling method can yield ineffective results. To account for this, the completion time behavior can be represented using a time-varying duration, $d_i(t)$. In this thesis, we account for time-varying processing time and characterize C_i using a task's completion time function $\delta(t)$, such that $C_i = \delta_i(s_i)$. Using this characterization, the time-varying duration can still be found, as $d_i(t) = \delta_i(t) - t$.

Definition 1 (Completion time function)**.** Given a time-horizon $H \in \mathbb{R}_{>0}$, a function $\delta : D \to \mathbb{R}$ is a completion time function if

$$\delta(t_1) \leq \delta(t_2) \quad \forall t_1, t_2 \in D \text{ with } t_1 \leq t_2 \tag{2.1}$$

$$\delta(t) \geq t \qquad \forall t \in D \tag{2.2}$$

where $D = [0, H]$.

Property (2.1) enforces that a task cannot be finished earlier if started later. In such a situation where a task might finish sooner if started later, one could wait until the later

starting time to actually start the task, in which case the completion time would be the same. Property (2.2) enforces that a task cannot be finished prior to when it is started.

Example 1. Consider a candidate function

$$\delta(t) = \frac{3}{2}t + \sin(t) + \frac{1}{2}\sin(2t) + 1 \tag{2.3}$$

with a prediction horizon of $H \geq 2\pi$. The candidate function is a valid completion time function as it obeys the two requisite properties: one, that it is monotonically non-decreasing,

$$\frac{d\delta}{dt}(t) = \frac{3}{2} + \cos(t) + \cos(2t) \geq 0 \tag{2.4}$$

for all $t \in D$, which can be seen with the equivalent statement

$$\begin{aligned} \cos(t) + 2\cos^2(t) + \frac{1}{2} &\geq 0 \\ \min_{t\in[0,2\pi]} \cos(t) + 2\cos^2(t) &\geq -\frac{1}{2} \\ \min_{t\in\{0,2\pi,\cos^{-1}(-\frac{1}{4})\}} \cos(t) + \cos^2(t) &\geq -\frac{1}{2} \\ -\frac{1}{8} &\geq -\frac{1}{2}, \end{aligned} \tag{2.5}$$

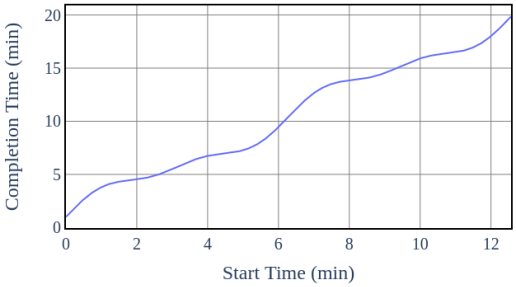
and two, that it is super-linear,

$$\frac{3}{2}t + \sin(t) + \frac{1}{2}\sin(2t) + 1 \geq t \tag{2.6}$$

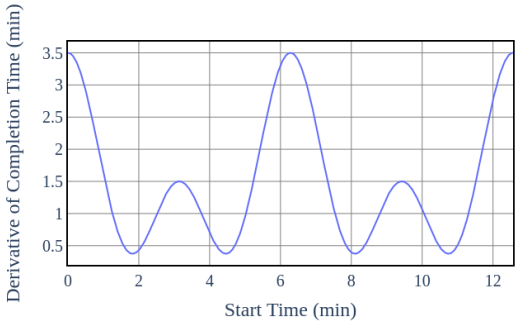
$$\tag{2.7}$$

for all $t \in D$, which can be seen with the equivalent statement

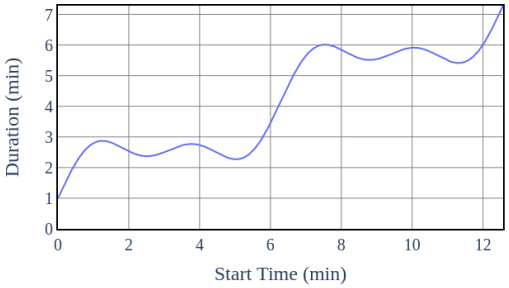
$$\begin{aligned} \min_{t \in [0, 2\pi]} \frac{1}{2} t + \sin(t) + \frac{1}{2} \sin(2t) &\geq -1 \\ 0 &\geq -1. \end{aligned} \tag{2.8}$$



(a) Completion Time Function



(b) Completion Time Function Derivative



(c) Completion Time Function Duration

Figure 2.1: Completion time graph with derivative and duration for (2.3). Note that the derivative is non-negative over the entire interval, and the duration is non-negative over the interval, as they relate to the completion time function properties.

For a problem in the base class with a completion time function, we can find the makespan by self-composing the completion time function about 0 as many times as there are tasks:

$$W_{min}(N) = \delta^N(0), \tag{2.9}$$

where $\delta^N(t)$ denotes the N^{th} self-composition of δ about t. For scheduling purposes, we need only consider problems in which the length of the scheduling window is at most the prediction horizon, that is $W \leq H$. Thus, in order for there to be a feasible schedule with any number of tasks, the completion time of a task begun at the start must be at most the prediction horizon, that is $\delta(0) \leq H$.

2.1 Over-approximation

In this context, an approximation is a function that can be used in place of the completion time function to solve for a schedule. Over particular interest to the scheduling problems discussed in this work is over-approximations.

Definition 2. A function $\widetilde{f}(x)$ is an over-approximation of $f(x)$ if

$$\widetilde{f}(x) \geq f(x) \tag{2.10}$$

for all x.

When scheduling using an approximation of the completion time function, an over-approximation is necessary in order to ensure that the resulting schedule is feasible when evaluated with the original completion time function. We will first give an intuitive understanding of this with an example, and then provide a proposition that embodies this statement and a proof for the proposition.

Example 2. Consider a problem with a prediction horizon $H = 4\pi$ and the example completion time function (2.3). Let δ_{approx} be an approximation of the completion time function determined by

$$\delta_{\text{approx}} = \frac{\delta(H) - \delta(0)}{H} t + \delta(0), \tag{2.11}$$

which for this example is

$$\delta_{\text{approx}} = \frac{3}{2}t + 1. \tag{2.12}$$

Let $\delta_{\text{over-approx}}$ be

$$\delta_{\text{over-approx}} = \frac{3}{2}t + \frac{5}{2}. \tag{2.13}$$

It can be seen that (2.13) is an over-approximation of (2.3), as

$$\delta_{\text{over-approx}}(t) = \frac{3}{2}t + \frac{5}{2} \geq \frac{3}{2}t + \sin(t) + \frac{1}{2}\sin 2t + 1 = \delta(t), \tag{2.14}$$

reduces to

$$\frac{3}{2} \geq \sin(t) + \frac{1}{2}\sin(2t) \tag{2.15}$$

which holds for all $t \in D$.

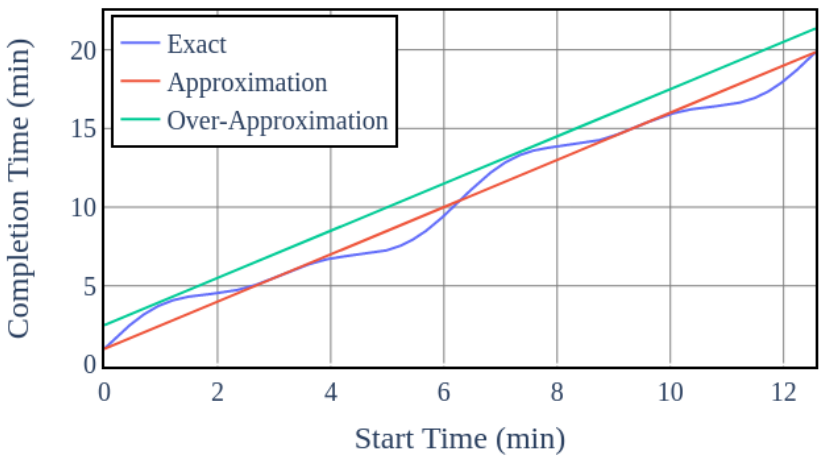


Figure 2.2: Completion time graph of (2.3) overlayed with the direct approximation (2.12) and the over-approximation (2.13).

Let E_{approx} and $E_{\text{over-approx}}$ be the total error of δ_{approx} and $\delta_{\text{over-approx}}$ over D. In terms of overall error, the over-approximation is less favorable, i.e. $E_{\text{approx}} < E_{\text{over-approx}}$, computed for this example as

$$\begin{aligned} E_{\text{approx}} &= \int_D |\delta_{\text{approx}}(t) - \delta(t)|dt \\ &= \int_0^{4\pi} |-\sin(t) - \frac{1}{2}\sin(2t)|dt \\ &= 8 \end{aligned} \tag{2.16}$$

and

$$\begin{aligned} E_{\text{over-approx}} &= \int_D |\delta_{\text{over-approx}}(t) - \delta(t)|dt \\ &= \int_0^{4\pi} |-\sin(t) - \frac{1}{2}\sin(2t) + \frac{3}{2}|dt \\ &= 6\pi \end{aligned} \tag{2.17}$$

respectively. The approximation makespan for N tasks is

$$\delta^N_{\text{approx}}(0) = \sum_{k=0}^{N-1} \frac{3}{2}, \tag{2.18}$$

so for $N = 3$ the approximation makespan is

$$\delta^3_{\text{approx}}(0) = 4.75, \tag{2.19}$$

which is less than the original completion time function makespan,

$$\delta(0) = 1 \tag{2.20}$$

$$\delta^2(0) = \frac{3}{2} + \sin(1) + \frac{1}{2}\sin(2) + 1 = 3.796 \tag{2.21}$$

$$\delta^3(0) = 6.56825. \tag{2.22}$$

This implies that if $N = 3$ and $W = 5$, the problem is infeasible using the original completion time function, but feasible using the direct approximation δ_{approx}, so any property of a schedule found using the direct approximation could not be guaranteed when the schedule is used with the original system. However, the over-approximation makespan is

$$\delta_{\text{over-approx}} = 11.875, \tag{2.23}$$

which is greater than the original completion time function makespan, and therefore a schedule found using the over-approximation can be guaranteed in the same sense that a schedule found using the direct approximation cannot.

Proposition 1. If $\widetilde{\delta}(t)$ is an over-approximation of a completion time function $\delta(t)$, i.e.

$$\widetilde{\delta}(t) \geq \delta(t) \tag{2.24}$$

for all $t \in D$, then any feasible problem in the base class using $\widetilde{\delta}(t)$ is also feasible using the underlying completion time function $\delta(t)$, i.e.

$$\widetilde{\delta}^N(0) \leq W \implies \delta^N(0) \leq W \tag{2.25}$$

for all $N \in \mathbb{Z}_{>0}$ and for all $W \in [0, H]$.

Proof. Note that for all scheduling windows, the makespan of an appoximation being less than the makespan of the completion time function is equivalent to the makespan of the approximation being less than the makespan of the completion time function, i.e.

$$(\widetilde{\delta}^N(0) \leq W \implies \delta^N(0) \leq W) \iff (\widetilde{\delta}^N(0) \geq \delta^N(0)). \tag{2.26}$$

So, if (2.26) holds, Proposition 1 will also hold, i.e.

$$\widetilde{\delta}(t) \geq \delta(t) \quad \forall t \in D \implies (\widetilde{\delta}^N(0) \leq \delta^N(0)) \quad \forall N \in \mathbb{Z}_{>0}. \tag{2.27}$$

We show (2.27), and thus Proposition 1, using a proof by induction. Assume that $\widetilde{\delta}(t)$ and $\delta(t)$ satisfy (2.24). We begin with the initial case, $N = 1$, and show (2.29) to be true. The first self-composition of each function about zero is

$$\widetilde{\delta}^1(0) \geq \delta^1(0), \tag{2.28}$$

which is equivalent to each function itself at zero,

$$\widetilde{\delta}(0) \geq \delta(0). \tag{2.29}$$

From (2.24) and the fact that $0 \in D = [0, H]$, we have know (2.29) to be true, which is what we set out to show.

Next, we assume (2.29) to hold for the n^{th} case, that is scheduling when $N = n$ tasks,

$$\widetilde{\delta}^n(0) \geq \delta^n(0), \tag{2.30}$$

and we show that it holds for the the $n+1^{\text{th}}$ case, that is when scheduling $N = n+1$ tasks,

$$\widetilde{\delta}^{n+1}(0) \geq \delta^{n+1}(0), \tag{2.31}$$

which is equivalent to the completion time of a task started at the completion time of the n^{th} task,

$$\widetilde{\delta}(\widetilde{\delta}^n(0)) \geq \delta(\delta^n(0)). \tag{2.32}$$

From the super-linear property of a valid completion time function (2.1) and the n^{th} case, we have that

$$\delta(\widetilde{\delta}^n(0)) \geq \delta(\delta^n(0)), \tag{2.33}$$

and from (2.24) and (2.33), we know that to be equivalent to

$$\widetilde{\delta}(\bar{\delta}^n(0)) \geq \delta(\widetilde{\delta}^n(0)), \tag{2.34}$$

and from (2.30) and (2.34), we know that to be equivalent to

$$\widetilde{\delta}(\bar{\delta}^n(0)) \geq \delta(\delta^n(0)), \tag{2.35}$$

which is equivalent to

$$\widetilde{\delta}^{n+1}(0) \geq \delta^{n+1}(0). \tag{2.36}$$

This completes the proof. □

2.2 Worst case processing time approximation

The worst case processing time (WCPT) approximation of a completion time function takes the form

$$\hat{\delta}(t) = t + d, \tag{2.37}$$

where d is the worst case processing time over the domain of interest,

$$d = \max_{t \in D} \delta(t) - t. \tag{2.38}$$

The WCPT approximation of the completion time function is an over-approximation and a valid completion time function over the approximation interval, which may be different from the scheduling window if the approximation interval is modified. Since the WCPT approximation is a valid completion time function, the makespan of problems in the base problem class sharing a common number of tasks N, can be found as follows:

$$\begin{aligned} \hat{W}_{min}(N) &= \hat{\delta}^N(0) \\ &= Nd. \end{aligned} \tag{2.39}$$

This is the form that must be used with existing scheduling methods, else the resulting schedule may not be feasible with the original completion time function.

2.3 Affine over-approximation

In the context of this work, an affine function takes the form

$$f(x) = ax + b. \tag{2.40}$$

An affine over-approximation $\bar{\delta}(t)$ of a completion time function $\delta(t)$ then takes the form

$$\bar{\delta}(t) = k_1 t + k_2 \geq \delta(t) \tag{2.41}$$

for all $t \in D$.

An affine over-approximation of a completion time function is appealing for three reasons:

1. As an over-approximation by definition, we can guarantee that any resulting schedules are feasible using the original completion time function, as shown by Proposition 1.

2. We can encode the affine over-approximation as in a mixed-integer program for

scheduling, with no corresponding increase in size of the program over using a WCPT approximation.

3. If we can find an affine over-approximation using our method, then the set of feasible problems using the WCPT approximation is a strict subset of the set of feasible problems using the affine over-approximation, shown by Proposition 2.

We present an algorithm and a quadratic program for finding such an affine over-approximation:

Algorithm 1 Algorithm for finding affine over-approximation of a continuous completion time function

input: Completion Time Function $\delta(t)$

output: Affine Over-Approximation k_1, k_2, Δ

initialize:

  ${}_{tmp}\Delta \leftarrow \Delta \leftarrow 0$

  ${}_{tmp}k_1 \leftarrow k_1 \leftarrow 1$

  ${}_{tmp}k_2 \leftarrow k_2 \leftarrow d$

while ${}_{tmp}k_2 \geq \delta(0)$ **and** $|\{t \mid \delta(t) - {}_{tmp}k_1 t - {}_{tmp}k_2 = 0, t \in D\}| = 0$ **do**

  $\Delta \leftarrow {}_{tmp}\Delta$

  $k_1 \leftarrow {}_{tmp}k_1$

  $k_2 \leftarrow {}_{tmp}k_2$

  ${}_{tmp}\Delta \leftarrow {}_{tmp}\Delta - i$

  ${}_{tmp}k_1 \leftarrow 1 + \frac{{}_{tmp}\Delta}{H}$

  ${}_{tmp}k_2 \leftarrow d - {}_{tmp}\Delta$

end while

Algorithm 1 iterates downwards from the WCPT approximation until it intersects the completion time function, remaining fixed at the intersection with the prediction horizon. The algorithm yields a defining metric of the approximation $\Delta \in (0, d)$, that can be used to uniquely define $\bar{\delta}(t)$.

If the completion time function is not known over the entire prediction horizon, but instead a sample is known, then the affine over-approximation can be found by solving a quadratic program with an objective of

$$\min_{k_1,k_2\geq 0} \quad \|\boldsymbol{e}\|_2^2, \tag{2.42}$$

where $\boldsymbol{e}$ is a vector of the error of the approximation at each sample time, encoded as

$$\boldsymbol{e} = \begin{bmatrix} \boldsymbol{t} & \mathbb{1} \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} - \boldsymbol{\delta}. \tag{2.43}$$

In order to ensure the resulting approximation is an over-approximation, we introduce a constraint on the error that is

$$\boldsymbol{e} \geq 0, \tag{2.44}$$

and in order to ensure that the resulting approximation returns an approximation that is over-bounded by the WCPT approximation, we add add the constraint

$$k_1 H + k_2 = H + d. \tag{2.45}$$

This quadratic program altogether together is

$$\begin{aligned} \min_{k_1,k_2\geq 0} \quad & \|\boldsymbol{e}\|_2^2 \\ \text{s.t.} \quad & \boldsymbol{e} = \begin{bmatrix} \boldsymbol{t} & \mathbb{1} \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} - \boldsymbol{\delta} \\ & \boldsymbol{e} \geq 0 \\ & k_1 H + k_2 = H + d. \end{aligned} \tag{2.46}$$

where $\boldsymbol{\delta}$ represents a vector of the samples, $\boldsymbol{t}$ represents a vector of the sample times, $\mathbb{1}$ is a vector of ones of the same length as the samples, H is the length of the approximation interval, and d is the worst case processing time. We assume that the completion time function sample includes a sample at the start, i.e. $\boldsymbol{\delta}_0 = \delta(0)$. When finding Δ using the output k_1, k_2 of the quadratic program (2.46), the range of Δ is the same as when it is found using Algorithm 1. The size of (2.46) is $T + 1$ constraints with 2 variables, where T is the number of sample times over the approximation interval.

2.3.1 Modification of the approximation interval

Observe that as the scheduling window will never be greater than the prediction horizon, there will not be a feasible schedule that includes a task starting time at or after the end of the prediction horizon. Knowing this, we can narrow the interval over which we make our approximation in order to gain a tighter approximation. When we do this, the interval becomes $[0, t_H]$, where

$$t_H = \min_{t \in D} t \tag{2.47}$$

$$\text{s.t.} \quad \delta(t) \geq H. \tag{2.48}$$

If we modify (2.43) and (2.45) correspondingly, we can generate approximations that are tighter over the relevant interval for scheduling. As the modified approximation interval is constructed such that no feasible schedule would have a starting time outside the interval, any schedules found using these approximations will still obey the same properties over the modified approximation interval, e.g. feasibility ordering due to over-approximation. Note that because a completion function can be monotonic non-decreasing, there can be multiple times that qualify for the new approximation interval. In order to produce beneficial results with our approximation algorithm, we take the earliest time at which the completion time function satisfies the requirement. Hereafter, this technique for the modification of the

approximation interval is applied for all examples and numerical studies.

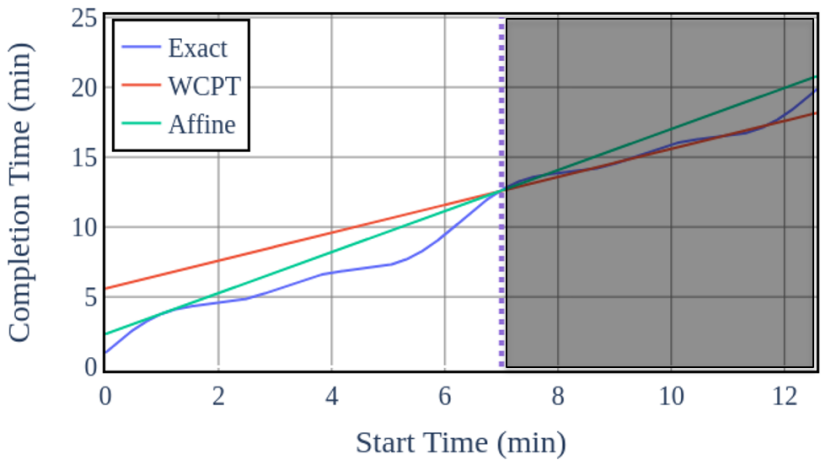


Figure 2.3: Resulting affine approximation and WCPT approximation of (2.3) using Algorithm 1. The modified approximation interval is represented by the unshaded area to the left of the dotted line. The end of the interval is indicated by the dotted purple line, which otherwise is indicated by the intercept of the WCPT approximation and the affine over-approximation.

2.3.2 Mixed-integer program

We solve for schedules of problems in the base problem class using a mixed-integer program. Specifically, those are problems involving finding starting times in a scheduling window for a set of tasks sharing a common completion time function and no additional constraints. The objective function of the program is

$$\min_{\boldsymbol{s},\boldsymbol{\sigma}} \max_{i} \quad C_i, \tag{2.49}$$

which corresponds to minimizing the makespan, i.e. the completion time of the last scheduled task. The starting times of each task are constrained to be non-negative,

$$s_i \geq 0 \quad \forall i \in N, \tag{2.50}$$

in order to enforce that all tasks start within the scheduling window. The no-overlap constraint is enforced using a disjunctive boolean decision variable, as described in [31], with a desired definition of

$$\sigma_{i,j} = \begin{cases} 1, \, s_j \leq C_i \\ 0, \, \text{otherwise} \end{cases} \tag{2.51}$$

which is enforced by the constraint

$$s_j - C_i - W(\sigma_{i,j} - 1) \geq 0 \quad \forall i \neq j \in N. \tag{2.52}$$

With the definition (2.51) for the disjunctive variable enforced by (2.52), the no-overlap constraint can be enforced with the program constraint

$$\sigma_{i,j} + \sigma_{j,i} = 1 \quad \forall i \neq j \in N, \tag{2.53}$$

which implies that for every pair of tasks, one and only one of the pair of tasks must complete before the other starts. When using the WCPT approximation, the completion time of a task is defined in the program by

$$C_i = s_i + d \quad \forall i \in N. \tag{2.54}$$

When using an affine over-approximation with coefficients k_1, k_2 such as one found using Algorithm 1, the completion time of a task is defined in the program by

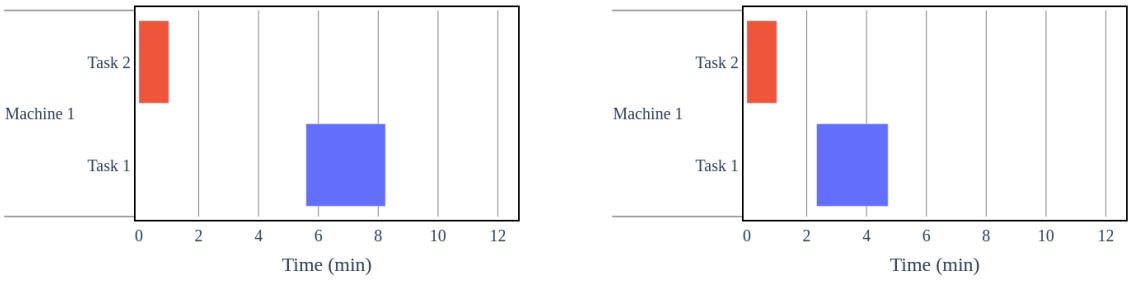
$$C_i = k_1 s_i + k_2 \quad \forall i \in N. \tag{2.55}$$

The scheduling window length is enforced by the constraint

$$C_i \leq W \quad \forall i \in N, \tag{2.56}$$

where W is the length of the scheduling window. This program altogether is

$$\begin{aligned} \min_{\boldsymbol{s},\boldsymbol{\sigma}} \max_{i} \quad & C_i \\ \text{s.t.} \quad & s_i \geq 0 \quad \forall i \in N \\ s_j - C_i - W(\sigma_{i,j} - 1) \geq 0 \quad & \forall i \neq j \in N \\ \sigma_{i,j} + \sigma_{j,i} = 1 \quad & \forall i \neq j \in N \\ C_i \leq W \quad & \forall i \in N \\ C_i = k_1 s_i + k_2 \quad & \forall i \in N. \end{aligned} \tag{2.57}$$



(a) WCPT Approximation Schedule

(b) Affine Approximation Schedule

Figure 2.4: The resulting schedule of program (2.57) when using the affine approximation and WCPT approximation of (2.3).

The resulting number of variables in program (2.57) is N^2 and the number of constraints is N^2, using either the WCPT approximation or affine over-approximation. Hence, the size of the program is unchanged when moving from the WCPT approximation to an affine over-approximation.

We do not assert that the mixed-integer program is the fastest way to solve for sched-

ules in the base problem class, however it provides us with a basis that we can use to directly compare the difficulty of scheduling by way of mixed-integer program using the two approximations.

2.3.3 Feasibility of scheduling using the affine over-approximation

We can represent the affine over-approximation produced using Algorithm 1 as

$$\bar{\delta}(t) = k_1 t + k_2, \tag{2.58}$$

with the coefficients as a function of the prediction horizon, the worst case processing time, and $\Delta \in [0, d)$,

$$k_1 = 1 + \frac{\Delta}{H} \tag{2.59}$$

$$k_2 = d - \Delta. \tag{2.60}$$

The affine over-approximation (2.58) is itself a valid completion time function when using coefficients (2.59) and (2.60) and a valid Δ, as found using Algorithm 1 or quadratic program (2.46). The resulting function will meet the monotonic non-decreasing property (2.1) of a valid completion time function, as the continuous completion time function property can be expressed as

$$\frac{d}{dt}\bar{\delta}(t) \geq 0, \tag{2.61}$$

which in terms of the affine over-approximation coefficients is

$$k_1 \geq 0, \tag{2.62}$$

and when evaluated in terms of Δ becomes

$$\frac{\Delta}{H} \geq 0. \tag{2.63}$$

We know (2.63) to be true as $H \in R_{>0}$ and $\Delta \in [0, d)$. The resulting function also meets the super-linear property (2.2) of a valid completion time function, which in terms of the affine over-approximation coefficients is

$$(k_1 - 1)t + k_2 \geq 0, \tag{2.64}$$

and when evaluated in terms of Δ becomes

$$\frac{\Delta}{H}t + d - \Delta \geq 0 \tag{2.65}$$

for all $t \in D$. We know (2.65) to be true, as $\Delta \in [0, d)$.

Since the resulting affine over-approximation is a valid completion time function, the makespan of scheduling N tasks using $\bar{\delta}(t)$, $\bar{W}_{min}(N)$, can be found as follows:

$$\begin{aligned} \bar{W}_{min}(N) &= \bar{\delta}^N(0) \\ &= k_2 \sum_{i=0}^{N-1} k_2^i \\ &= (d - \Delta) \sum_{i=0}^{N-1} (1 + \frac{\Delta}{H})^i. \end{aligned} \tag{2.66}$$

The WCPT approximation, $\hat{\delta}(t)$ is an over-approximation of the affine over-approximation, $\bar{\delta}(t)$ within the prediction horizon. That is

$$\bar{\delta}(t) = (1 + \frac{\Delta}{H})t + d - \Delta \leq t + d = \hat{\delta}(t) \tag{2.67}$$

for all $t \in D$, which is equivalent to:

$$\Delta(\frac{t}{H} - 1) \leq 0 \tag{2.68}$$

which we know (2.68) to be true, as $\Delta \geq 0$ and $t \in [0, H]$.

Proposition 2. If an affine over-approximation $\bar{\delta}(t)$ can be found with $\Delta > 0$, the set of feasible problems using $\hat{\delta}(t)$ is a strict subset of the set of feasible problems using $\bar{\delta}(t)$ in the base problem class. This means that the $\bar{\delta}(t)$ makespan is at most the $\hat{\delta}(t)$ makespan for every number of tasks, and that there exists a number of tasks for which the $\bar{\delta}(t)$ makespan is strictly less than the $\hat{\delta}(t)$ makespan, i.e.

$$(\bar{W}_{min}(N) \leq \hat{W}_{min}(N) \quad \forall N \in \mathbb{Z}_{>0}) \wedge (\exists N | \bar{W}_{min}(N) < \hat{W}_{min}(N)). \tag{2.69}$$

.

Proof. Assume that

$$\Delta > 0. \tag{2.70}$$

We know from (2.67) and Proposition 1, that

$$\bar{W}_{min}(N) \leq \hat{W}_{min}(N) \quad \forall N \in \mathbb{Z}_{>0}. \tag{2.71}$$

This leaves us to show that

$$\exists N | \bar{W}_{min}(N) < \hat{W}_{min}(N). \tag{2.72}$$

Consider the set of problems where $N = 1$, which is a subset of any non-empty set of

feasible problems. The equivalent of (2.72) for the case of $N = 1$ is

$$\bar{W}_{min}(1) < \hat{W}_{min}(1). \tag{2.73}$$

From (2.39) and (2.66), we have that (2.73) is equivalent to

$$d - \Delta < d, \tag{2.74}$$

which reduces to

$$\Delta > 0. \tag{2.75}$$

This completes the proof. □

2.4 Numerical results

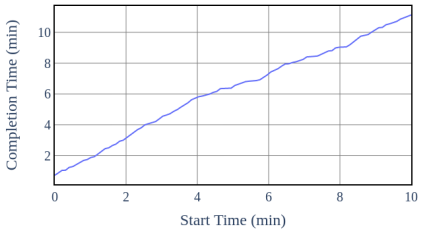
We demonstrate what we have shown using five sampled randomly generated completion time functions, which can be seen in Figures 2.5, 2.6, 2.7, 2.8, and 2.9 and are hereafter referred to as the demonstration functions. Each of the sampled functions, $\delta_n, n \in \{1, 2, 3, 4, 5\}$ is generated over a prediction horizon of $H = 10$ with one hundred samples. The zero-intercept of each is drawn from a uniform distribution over $[0.25, 1.25]$. Each successive sample difference is drawn from a uniform distribution over $[\max\{t_i - \delta_n[i - 1], 0\}, \phi_n t_i]$ where $\phi = [2, 2.5, 3, 3.5, 4]$, t_i is the sample time, and $\delta_n[i - 1]$ is the sample of δ_n at the previous time step. We then find the worst case processing time and affine over-approximation coefficients for each demonstration function over a reduced approximation interval. For a representation of each demonstration function, we have calculated the least squares linear fit of each demonstration function. The resulting linear fit, worst case processing time, and affine over-approximation coefficients are listed in Table 2.1. Note that for the first demonstration function, the affine over-approximation is equivalent

to the WCPT. As the slope of the linear fit increases, so does the difference between the WCPT and the affine over-approximation.

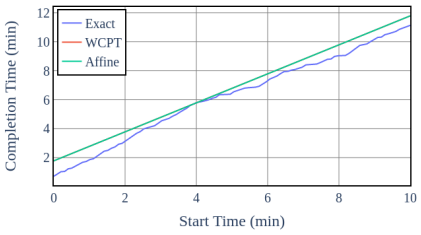
We use the mixed-integer program (2.57) to solve for the schedule using the demonstration function, the WCPT approximation of the demonstration function and the affine over-approximation of the demonstration function found using the quadratic program (2.46). The program solution is computed using the Python MIP package with CBC as the backed solver on an Intel Core i7-8700K clocked at 3.70 GHz [32][33]. As stated in Proposition 2, every problem in which the WCPT approximation is feasible, the affine over-approximation is also feasible, and the makespan of the affine over-approximation is at most that of the WCPT approximation.

Table 2.1: Function identifier, linear approximation, WCPT approximation, and affine over-approximation of demonstration functions.

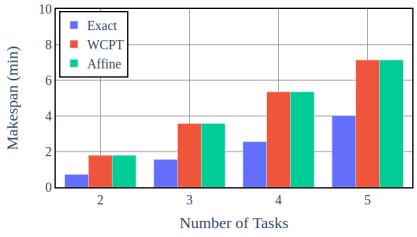
Function	h_1	h_2	d	k_1	k_2
1	1.009	1.195	1.785	1.0	1.785
2	1.168	0.990	2.660	1.103	1.630
3	1.511	1.349	6.357	1.210	4.261
4	1.664	0.6446	7.436	1.500	2.431
5	2.160	0.338	11.582	1.728	4.299



(a) Completion Time Function

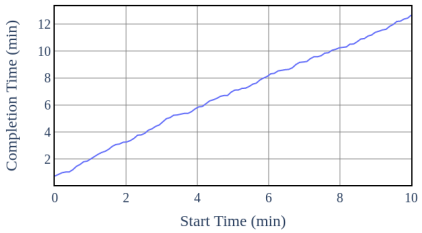


(b) Approximations

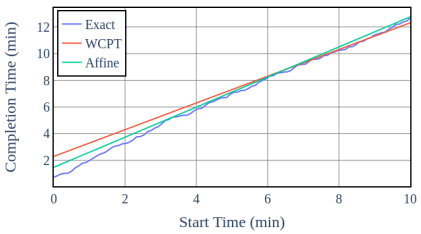


(c) Makespans

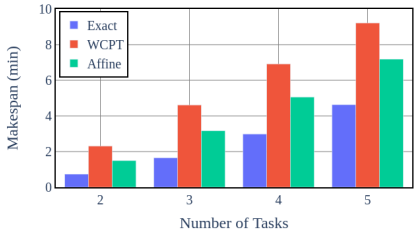
Figure 2.5: Completion time graph, approximations, and resulting makespans of demonstration function one. The modification of the approximation interval yields no different result from approximating over the original interval. Also, the WCPT approximation and the affine over-approximation resulting from quadratic program (2.46) are the same.



(a) Completion Time Function

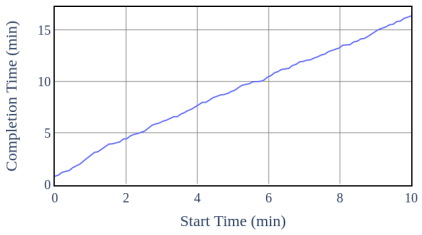


(b) Approximations

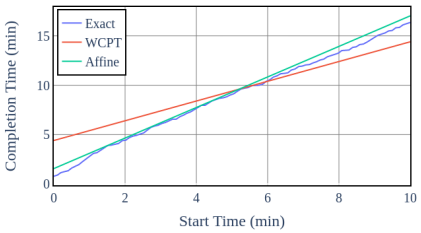


(c) Makespans

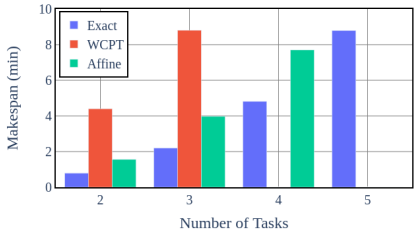
Figure 2.6: Completion time graph, approximations, and resulting makespans of demonstration function two. The modified approximation interval can be seen to end shortly before the intercept of the completion time function with the scheduling window length.



(a) Completion Time Function

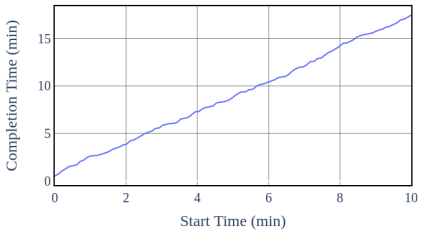


(b) Approximations

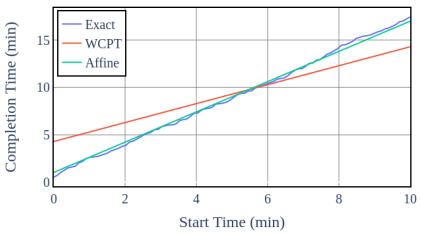


(c) Makespans

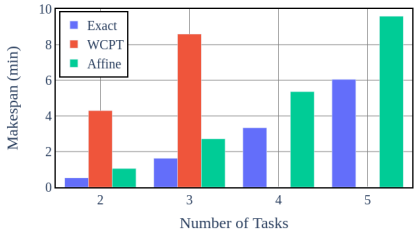
Figure 2.7: Completion time graph, approximations, and resulting makespans of demonstration function three. The modified approximation interval can be seen to end just before the intercept of the completion time function with the scheduling window length.



(a) Completion Time Function

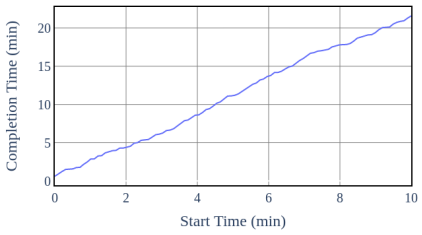


(b) Approximations

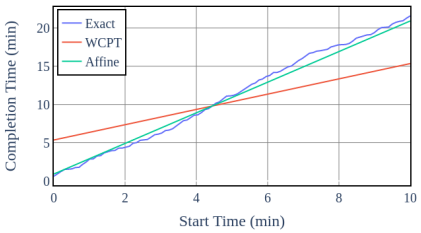


(c) Makespans

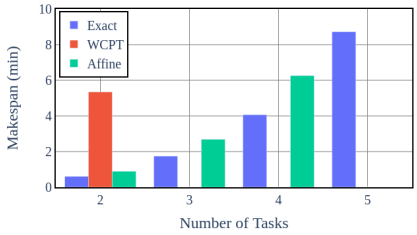
Figure 2.8: Completion time graph, approximations, and resulting makespans of demonstration function four. The modified approximation interval can be seen to end at the intercept of the completion time function with the scheduling window length.



(a) Completion Time Function



(b) Approximations



(c) Makespans

Figure 2.9: Completion time graph, approximations, and resulting makespans of demonstration function five. The modified approximation interval can be seen to end at the intercept of the completion time function with the scheduling window length.

CHAPTER 3
SCHEDULING CONSTRAINTS

In order to make practical use of the affine over-approximation, we must examine other types of scheduling problems. We do not show the same results as we did for the base problem class, but instead demonstrate numerically that for certain problems within these additional classes, the results of scheduling using the affine over-approximation provides similar behavior to that shown for the base problem class in Chapter 2.

In this chapter, we review variations of the base scheduling problem that capture different behaviors and requirements. Except for machine precedence, all variations originate from Graham's taxonomy. We have only included those that we need to solve for a schedule in our case study. Others that would materially affect the efficiency of the approximation method include release times, machines whose performance is proportional to each other, and locality requirements.

3.1 Nonuniform tasks

When a problem involves tasks that do not have identical completion time characteristics, the problem has nonuniform tasks. The scheduling of two applications composed of a different number of atomic operations is an example of a problem with nonuniform tasks. The solutions to problems with nonuniform tasks modelled using the completion time representation are nontrivial, unlike the solutions to problems with nonuniform tasks modelled using processing time. In effect, finding an exact solution to the problem becomes intractable, which allows us to demonstrate the benefit of using an approximation.

3.1.1 Modified MIP

For nonuniform problems, we modify the completion time constraint for the WCPT approximation in program (2.57) to be

$$C_i = s_i + d_i \tag{3.1}$$

for all $i \in N$, where d_i is the worst case processing time for task i. We modify completion time constraint for the affine approximation to be

$$C_i = k_{i,1} s_i + k_{i,2} \tag{3.2}$$

for all $i \in N$, where $k_{i,1}, k_{i,2}$ are the affine approximation coefficients for task i.

The number of variables of the mixed-integer program used to solve a problem with nonuniform task completion time functions is the same as the number of variables and constraints of the mixed-integer program (2.57). This is to say that the difficulty of the programs scales the same, although the actual time to compute a problem of a certain size can differ. However, the approximation algorithm must now be computed for each task, so the complexity scales linearly with the number of tasks. The basic problem class can be reduced to a problem with nonuniform completion times by creating N copies of the relevant approximation parameters.

3.1.2 Benefits of approximation

Let us introduce mixed-integer program (3.3) that solves for the schedule of a nonuniform problem using an exact sample of the completion time function. We must modify the sampled function to solve for the schedule, changing $\delta : T \to \mathbb{R}_{>0}$ into $\check{\delta} : T \to Y$ where T represents the sampled input space, and Y represents a sampled version of the output space using the same sampling technique as the input space. We solve this by rounding up

the output to the nearest sample point. Note that this results in a slight over-approximation of the sampled function, so the results from Proposition 1 hold for $\check{\delta}$. Scheduling using the sampled exact version of the completion time function becomes an assignment problem, where each task is now assigned to a starting time-step. The program for finding a schedule of nonuniform tasks using an exact sample of the completion time function is as follows:

$$\min_{x,\sigma} \max_{i} \quad C_i \tag{3.3}$$

$$\text{s.t.} \quad \sigma_{i,j} \in \{0,1\} \quad \forall i,j \in N \tag{3.4}$$

$$x_{i,j} \in \{0,1\} \quad \forall i \in N, j \in t \tag{3.5}$$

$$\sum_{j=1}^{T} X_{i,j} = 1 \quad \forall i \in N \tag{3.6}$$

$$s_i = \sum_{j=1}^{T} x_{i,j} t_j \quad \forall i \in N \tag{3.7}$$

$$C_i = \sum_{j=1}^{T} x_{i,j} \check{\delta}_{i,j} \quad \forall j \in N \tag{3.8}$$

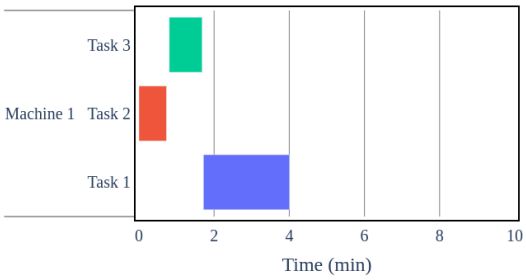
$$s_i - C_i - W(\sigma_{i,j} - 1) \geq 0 \quad \forall i,j \in N \tag{3.9}$$

$$\sigma_{i,j} + \sigma_{j,i} = 1 \quad \forall i,j \in N. \tag{3.10}$$

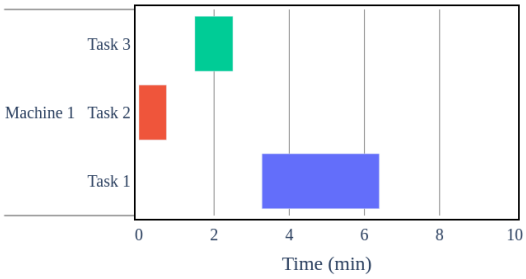
Program (3.3) has $N(N-1+T)$ variables, and $N(N+2)$ constraints. Thus, the exact program is larger than the approximate program if $T \geq N$. It is reasonable to assume $T \geq N$, as there would be no feasible schedule using the disjunctive constraints if there were fewer times to start a task on than there were tasks.

3.1.3 Example

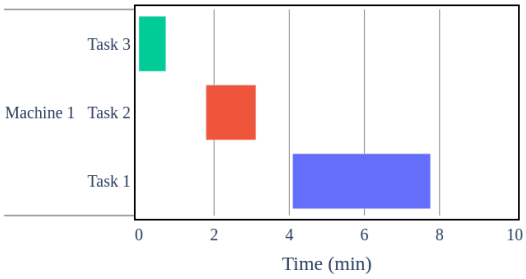
Figure 3.1 shows the resulting schedules of the program (3.3) and the program (2.57) with the addition of constraint (3.2) for a problem composed of the first three demonstration functions as seen in Table 2.1. As not all of the demonstration functions are equivalent over the entire prediction horizon, scheduling three tasks whose completion times are described by the first three demonstration functions is a problem with nonuniform tasks.



(a) Exact Schedule



(b) Approximation Schedule



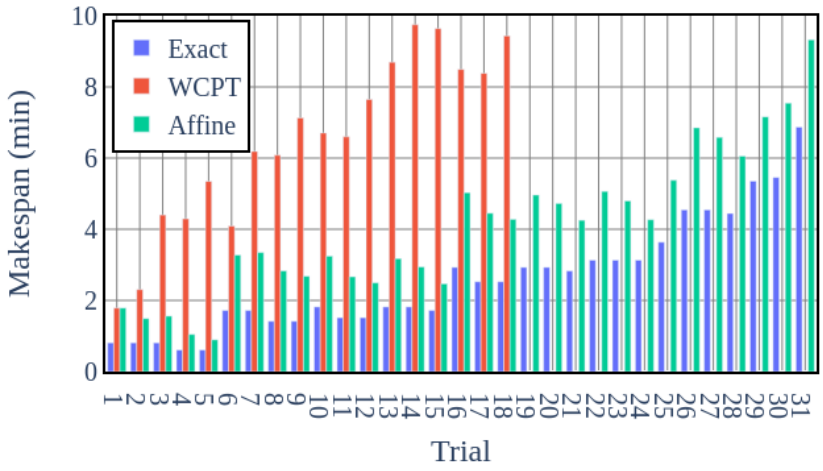
(c) WCPT Schedule

Figure 3.1: The resulting schedules and assignments for the example of scheduling three nonuniform tasks described by the first three demonstration functions.

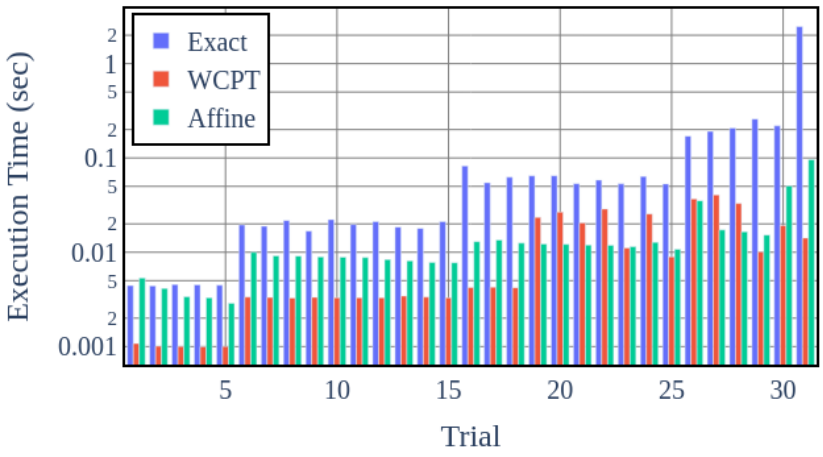
3.1.4 Numerical results

Figure 3.9a shows the resulting makespans of scheduling elements of the powerset of the set of demonstration functions. Figure 3.9b shows the time taken to compute the schedules, and find both the WCPT approximation and the affine over-approximation. These two

figures show the trade-off between the resulting scheduling feasibility and time to compute the schedule over the three methods. Table 3.1 lists the task set combinations associated with the trials in the figures.



(a) Minimum Makespans



(b) Execution Times

Figure 3.2: A comparison of the resulting execution times of the scheduling methods, and their resulting makespans for problems with nonuniform tasks.

Table 3.1: Task set combinations and their associated trial identifier for the numerical results of scheduling nonuniform tasks.

Index	1	2	3	4	5	6	7	8	9	10	11
Trial	1	2	3	4	5	1,2	1,3	1,4	1,5	2,3	2,4
Trial	12	13	14	15	16	17	18	19	20	21	22
Task Set	2,5	3,4	4,5	1,2,3	1,2,4	1,2,5	1,3,4	1,3,5	1,4,5	2,3,4	2,3,5

3.2 Task precedence

When a problem involves tasks that must complete in a partial order, the problem features precedence relations between tasks. The scheduling of a job that can be broken down into constituent series components is an example of a problem with precedence relations. For our purposes, the precedence relations are thought of as a graph defined by the matrix $A \in \{0,1\}^{NxN}$, where $a_{i,j}$ indicates the existence of an directed edge from vertex i to vertex j, each of which represents that task. An edge between task i and task j indicates that the completion time of task i must be at most the starting time of task j in any valid schedule. In order for any scheduling problem with a given precedence graph to be feasible, the resulting graph must not have any cycles, which includes self-loops. In these requirements are met, the precedence graph is a directed acyclic graph (DAG).

3.2.1 Modified MIP

For problems with precedence, we introduce a further constraint on the disjunctive variable $\sigma_{i,j}$:

$$\sigma_{i,j} \geq a_{i,j} \tag{3.11}$$

for all $i \neq j$ and $i, j \in N$. In order to reduce a problem in the base problem class to one with precedence constraints, the problem is treated as having no precedence relations between tasks, $a_{i,j} = 0$ for all $i, j \in N$. The precedence relations between tasks adds a further $N(N-1)$ constraints.

3.2.2 Example

Figure 3.4 shows the resulting schedules from a problem consisting of the first three demonstration functions as seen in Table 2.1, and precedence relations between the tasks determined by a graph show in Figure 3.3.

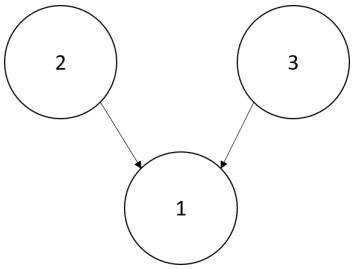
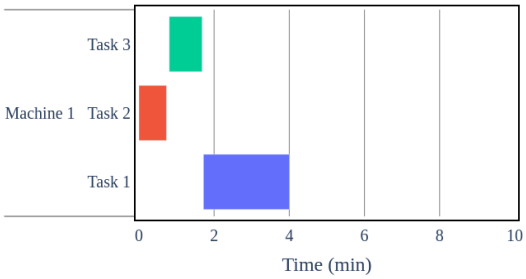
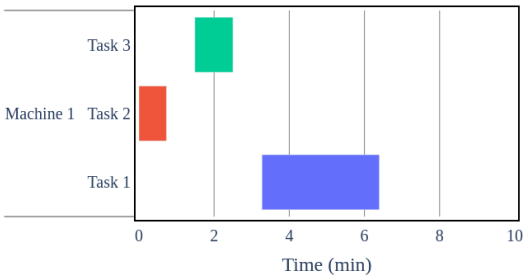


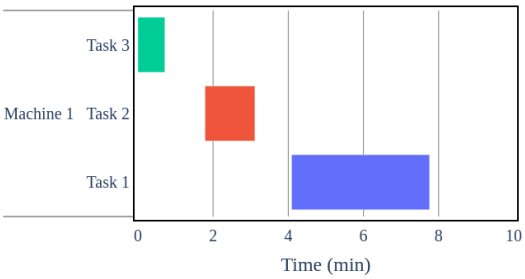
Figure 3.3: Precedence graph for the example of a problem with task precedence.



(a) Exact Schedule



(b) Approximation Schedule



(c) WCPT Schedule

Figure 3.4: The resulting schedules and assignments for the example of scheduling three nonuniform tasks described by the first three demonstration functions and precedence constraints described by the graph in Figure 3.3.

3.2.3 Numerical results

A collection of maskepans and execution times for problems with precedence are presented in Figure 3.5. These values were generated using the powerset of the demonstration functions, and for each element of the powerset, we created a set of problems consisting of all possible DAGs of the member demonstration functions. Only the problems that are feasible using the affine over-approximation are shown.

3.3 Multiple machines

When a problem involves more than one machine that can be used to complete tasks, the problem features multiple machines. In cases where the machines are indistinguishable in their completion time characteristics, the machines are uniform. The scheduling and assignment of application components to identical processors that are configured identically is an example of a problem with uniform machines.

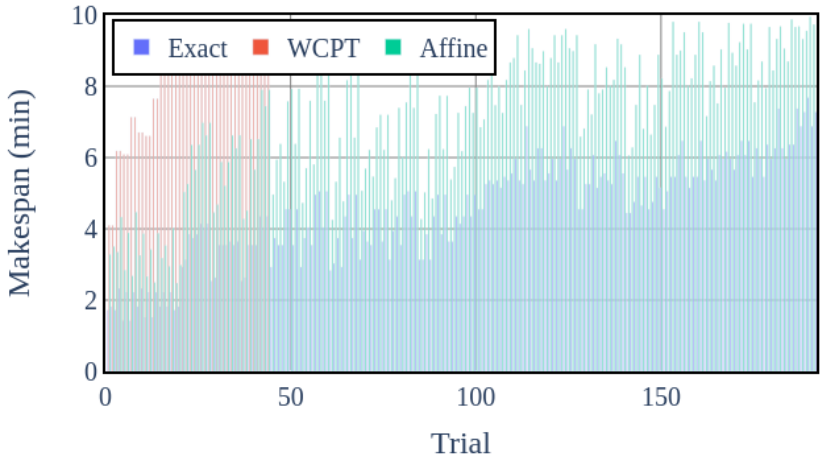
3.3.1 Modified MIP

For problems with multiple machines, we introduce introduce the processor assignment enumeration for task i,

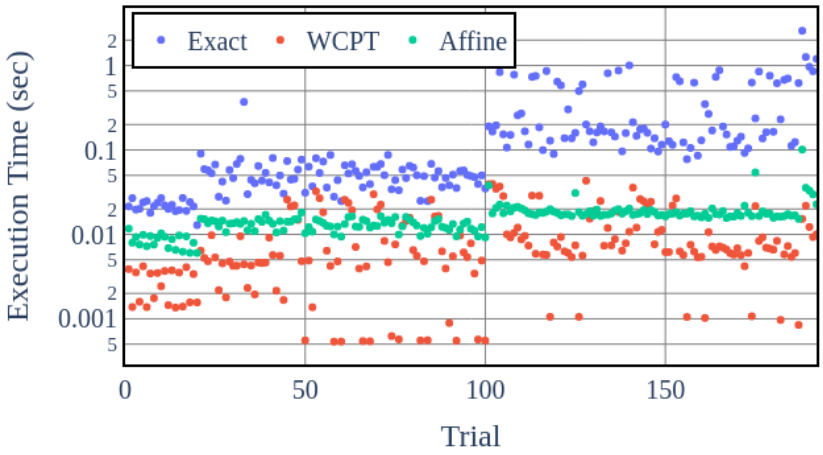
$$p_i \in M \tag{3.12}$$

for all $i \in N$. For problems with multiple machines, we introduce an assignment variable $x_{j,k}$ and another disjunctive variable $\epsilon_{i,j}$, defined in the program by

$$x_{i,k} = \begin{cases} 1, \text{ if task } i \text{ is assigned to machine } k \\ 0, \text{ otherwise} \end{cases} \tag{3.13}$$



(a) Minimum Makespans



(b) Execution Times

Figure 3.5: A comparison of the resulting execution times of the scheduling methods, and their resulting minimum makespans for problems with task precedence.

Using $x_{i,k}$, we define p_i in the modified program by

$$p_i = \sum_{k=1}^{M} k x_{i,k} \quad \forall j \in N. \tag{3.14}$$

In order to enforce a machine-specific version of the no-overlap constraint, we introduce an addition disjunctive boolean decision variable

$$\epsilon_{i,j} = \begin{cases} 1, \, p_i < p_j \\ 0, \, \text{otherwise} \end{cases} \tag{3.15}$$

As tasks can now overlap as long as they are not on the same machine, we need to modify the original disjunctive constraint (2.53) to an inequality,

$$\sigma_{i,j} + \sigma_{j,i} \leq 1 \quad \forall i \neq j \in N, \tag{3.16}$$

and a similar constraint for the machine disjunctive variable,

$$\epsilon_{i,j} + \epsilon_{j,i} \leq 1 \quad \forall i, j \in N, i \neq j, \tag{3.17}$$

which when combined with

$$\epsilon_{i,j} + \epsilon_{j,i} + \sigma_{i,j} + \sigma_{j,i} \geq 1 \quad \forall i, j \in N, i \neq j, \tag{3.18}$$

enforces the machine specific version of the no-overlap constraint.

In order for this constraint to work as desired, we need to enforce the definition of the machine disjunctive variable, which is done with the constraints

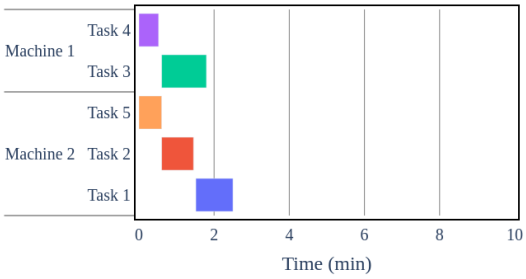
$$p_j - p_i - \epsilon_{i,j} M \leq 0 \quad \forall i, j \in N, i \neq j \tag{3.19}$$

$$p_j - p_i - 1 - (\epsilon_{i,j} - 1) M \geq 0 \quad \forall i, j \in N, i \neq j. \tag{3.20}$$

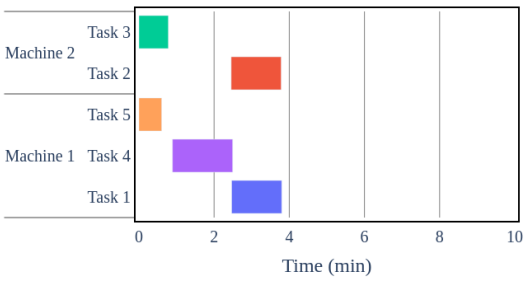
The assignment of tasks to multiple machines introduces $N(N + M)$ additional variables, and $3N(N - 1) + N$ additional constraints. In order to reduce a problem in the base class to one with multiple machines, set $M = 1$.

3.3.2 Example

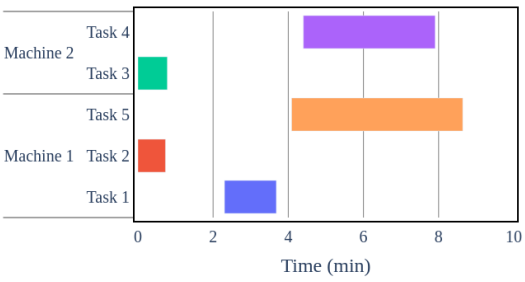
For an example of scheduling with uniform machines, consider the case where there are two machines available to schedule a set of nonuniform tasks composed of all five of the demonstration functions as seen in Table 2.1. The resulting schedules can be seen in Figure 3.6.



(a) Exact Schedule



(b) Approximation Schedule



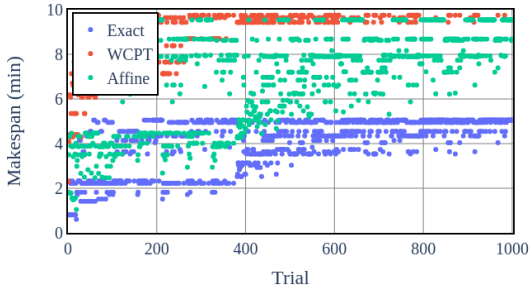
(c) WCPT Schedule

Figure 3.6: The resulting schedules and assignments for the example of scheduling the five demonstrations functions as nonuniform tasks onto two machines.

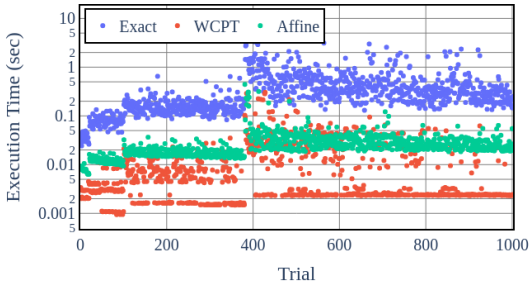
3.3.3 Numerical results

Figure 3.7 shows a comparison of makespans and execution times using the exact formulation, WCPT approximation, and affine over-approximation for multiple machines. These results are computed for the same set of problems as the precedence relation numerical results, but with two, three, and four machines available, as seen in the first, second,

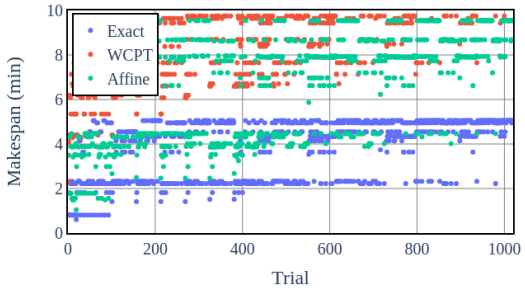
and third rows respectively.



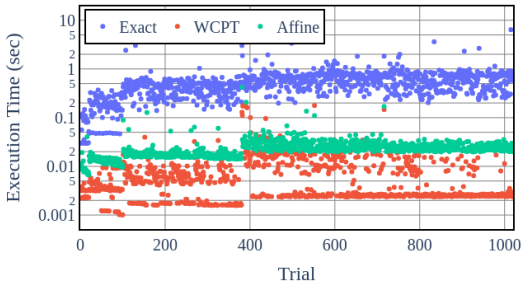
(a) Minimum Makespans



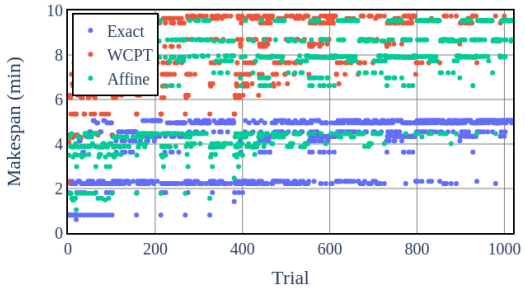
(b) Execution Times



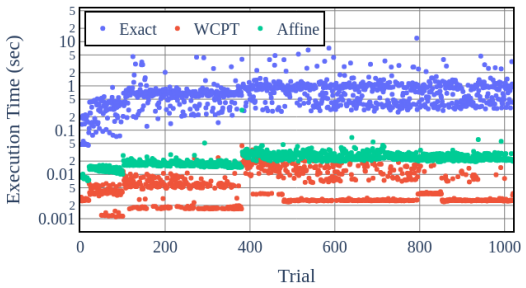
(c) Minimum Makespans



(d) Execution Times



(e) Minimum Makespans



(f) Execution Times

Figure 3.7: A comparison of the resulting execution times of the scheduling methods, and their resulting minimum makespans for problems with two, three, and four machines in descending order.

3.4 Nonuniform machines

When the problem involves multiple machines that provide different completion time behavior for the same task, the problem features nonuniform machines. A processor scheduling example would consist of multiple identical processors available that were clocked at different rates. If the problem features nonuniform machines, we need to modify the completion time equation for each task within the program. The completion time of a given task becomes a function of both the starting time and the machine assignment of the task. The immediate modification would be to solve for an affine over-approximation for every completion time function unique to task and machine, $\delta_{i,k}(t)$ for task i on machine k, and then introduce the constraint

$$C_i = k_{i,p_i,1} s_i + k_{i,p_i,2}, \tag{3.21}$$

where $k_{i,p_i,1}, k_{i,p_i,2}$ are the first and second coefficients, respectively, of the affine approximation for task i and its assigned machine p_i as defined in (3.17). However, this constraint on the completion time is non-convex, and thus not solvable using the mixed-integer program in its current form. We introduce three alternative methods to solve the problem with nonuniform machines.

3.4.1 Method 1

In the first method, we find one affine approximation over all machines for each task. The mixed-integer programming completion time constraint then takes the form of

$$C_i = k_{i,1} s_i + k_{i,2} p_i + k_{i,3}. \tag{3.22}$$

This is the simplest method in terms of the MIP program size. The error of this approximation is dependent upon the ordering of the tasks, which we can change without

materially affecting the outcome, but the ordering must be shared amongst all tasks. Finding such an ordering requires searching through all permutations of the machine ordering. Finding the optimal ordering becomes intractable with large numbers of machines, so we use a heuristic. We first find N candidate orderings, one for each task. The candidate ordering is determined by sorting the worst case processing times of that task on each machine. We then solve for the ordering that yields approximations for each task with the lowest total error. This method finds the optimal ordering from amongst N possible orderings, as opposed to $M!$ possible orderings.

3.4.2 Method 2

In the second method, we find one slope for all machines, but a separate intercept. The mixed-integer program completion time constraint then takes the form of

$$C_i = k_{i,1} s_i + \sum_{j=1}^{M} k_{i,j} x_{i,j}. \tag{3.23}$$

3.4.3 Method 3

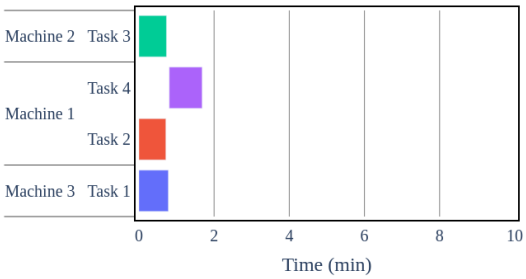
In the third method, we solve the entire problem sequentially, first solving for the schedule of all possible assignments, and then searching for the best amongst those schedules. The mixed-integer program completion time constraint then takes the form as stated in (3.21). However, the overall problem now takes the form of

$$\min_{p} \min_{s} \max_{i} C_i, \tag{3.24}$$

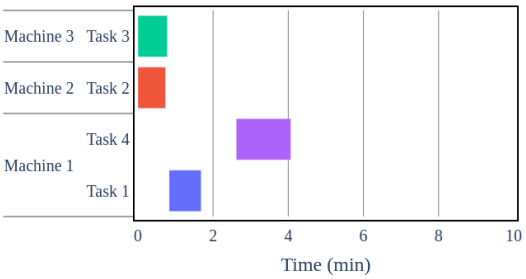
where p is the assignment vector of machines. Unless otherwise constrained, the top level optimization is a combinatorial search problem of size M^N.

3.4.4 Example

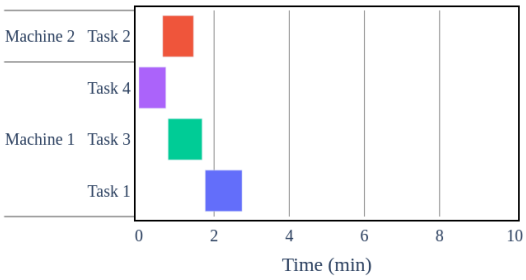
Consider the case of scheduling four uniform tasks on three machines whose behaviors are characterized by the first three demonstration functions as seen in Table 2.1. The resulting schedules and approximations using each method can be seen in Figure 3.8.



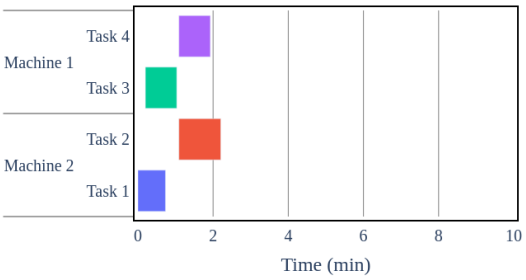
(a) Exact Schedule



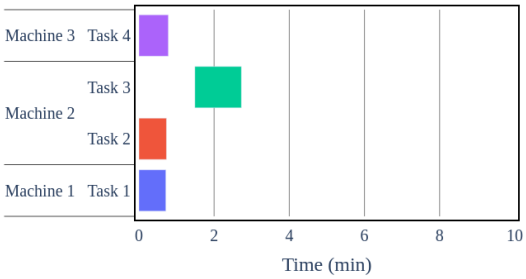
(b) WCPT Schedule



(c) Method 1 Schedule



(d) Method 2 Schedule



(e) Method 3 Schedule

Figure 3.8: The resulting schedules and assignments for the example of scheduling four uniform tasks onto three nonuniform machines as described by the first three demonstration functions.

3.4.5 Numerical results

In a similar manner to the nonuniform task completion time numerical results, we present in Figure 3.9 a comparison of the three methods for approximating nonuniform machines, the WCPT method, and the exact method. The problem set over which these are computed is the powerset of nonuniform machines, with five uniform tasks. However, the five completion time functions are associated with machines as opposed to tasks, and all tasks have the same completion time function for a given machine.

3.5 Heterogeneous machines

When a problem involves tasks that can only be assigned to a strict subset of the machines available, the problem has heterogeneous machines.

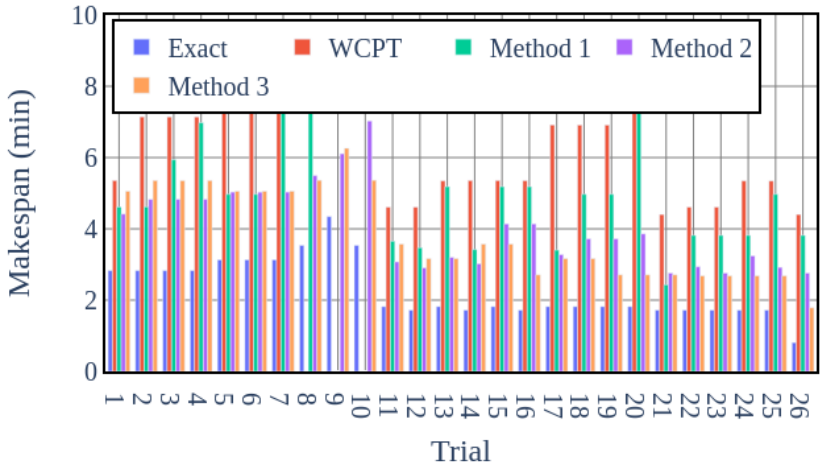
3.5.1 Modified MIP

If a problem features heterogeneous machines, determined by a typing tensor U, where $u_{i,k}$ indicates that job i can be assigned to machine k, we introduce the constraint

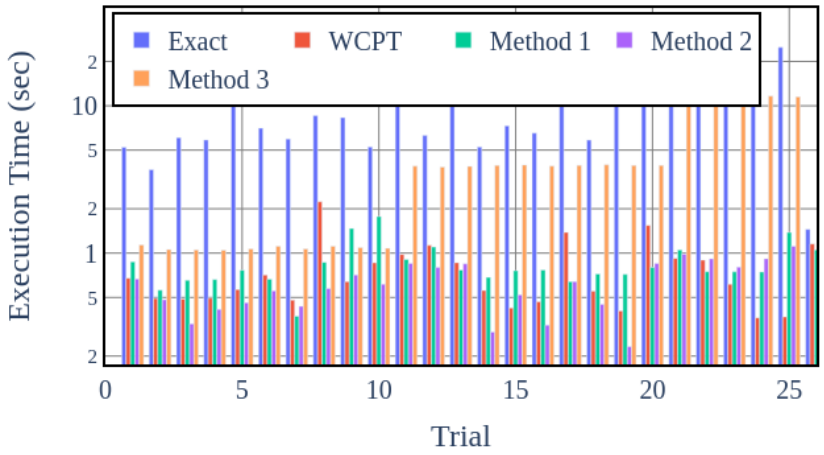
$$x_{i,k} \leq u_{i,k} \quad \forall i \in N, k \in M. \tag{3.25}$$

When a problem has machines that are both nonuniform and heterogeneous, the constraint (3.25) can be used in combination with (3.24) to reduce the size of the of the search set for the outer optimization problems. There are also less approximations per task that need to be found, as only the approximations on machines of the same types as a task are relevant.

Another method to model heterogeneous machines is by treating the completion time for a task on a machine of a different type to be either infinite or the prediction horizon with an infinitesimal increment. However, this does not allow us to constrain the problem set when using Method 3.



(a) Minimum Makespans



(b) Execution Times

Figure 3.9: A comparison of the resulting execution times of the scheduling methods, and their resulting minimum makespans for problems with multiple nonuniform machines.

3.6 Machine precedence

When the problem involves multiple machines and a limitation on what machines a task can be assigned to based on what machines other tasks were assigned to and the precedence relations between tasks, the problem involves machine precedence. The scheduling and

assignment of processing and communication tasks on a grid of processors is an example of a problem with machine precedence. In this example, consider the need to communicate processing task results between processors. The communication of a processing task's results can be modelled as a task itself, but can only be assigned to links adjacent to the processor the processing task is assigned to. This requirement is a machine precedence relation, where the processor precedes the communication link. Unlike the task precedence graph, the machine precedence graph can posses cycles as well as self-edges.

3.6.1 Modified MIP

If the problem features machine precedence, given in the form of an adjacency graph $B \in \{0,1\}^{MxM}$, where $b_{i,j}$ indicates that machine i precedes machine j, we first introduce the boolean decision variable

$$\gamma_{i,j,h,k} = \begin{cases} 1, \text{ task } i \text{ is assigned to machine } h \text{ and task } j \text{ is assigned to machine } k \\ 0, \text{ otherwise} \end{cases} \tag{3.26}$$

which can be thought of as whether or not an edge shared between the two machines would be utilized if the two tasks required it. The definition of this new variable is enforced with the constraints

$$x_{i,h} - \gamma_{i,j,h,k} \geq 0 \quad \forall i,j \in N, i \neq j, h,k \in M \tag{3.27}$$

$$x_{j,k} - \gamma_{i,j,h,k} \geq 0 \quad \forall i,j \in N, i \neq j, h,k \in M \tag{3.28}$$

$$x_{i,h} + x_{j,k} - 1 - \gamma_{i,j,h,k} \leq 0 \quad \forall i,j \in N, h,k \in M. \tag{3.29}$$

A similar version of this decision variable formulation was first used in [21] for consideration of active communications links for consideration of communication delays. The new decision variable can then be used to ensure that the resulting assignment matches the

requirements for the task precedence using the given machine precedence, enforced by the constraint

$$a_{i,j}\gamma_{i,j,h,k} \leq b_{h,k} \quad \forall i,j \in N, h,k \in M. \tag{3.30}$$

Machine precedence introduces N^2M^2 variables and $4N^2M^2$ constraints. A problem with multiple machines can be reduced to a problem with machine precedence by making all machines precede all other machines, $b_{h,k} = 1$ for all $h,k \in M$. Note that this is similar to reducing a problem in the base problem class to one with precedence between tasks.

CHAPTER 4

CASE STUDY: SATELLITE CONSTELLATION

For a case study, consider a constellation of satellites orbiting a celestial body composed of multiple orbital planes that have an application to process amongst the onboard processors of the constellation. An example of such a constellation is SpaceX's Starlink, a communication constellation speculated to have inter-satellite links used to provide broadband internet access across the Earth [34].

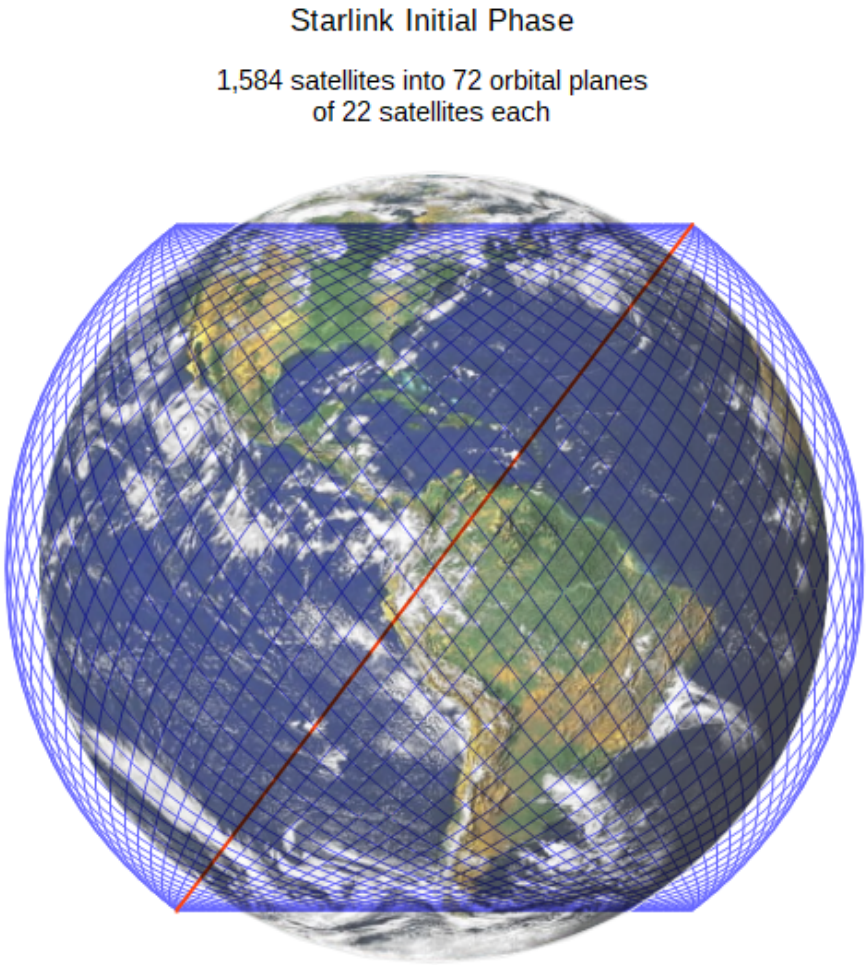


Figure 4.1: Orbital planes of the Starlink initial phase. Figure is publicly available on Wikipedia[35].

Let us instead consider a constellation of Earth Observation Satellites (EOS) that are searching for particular features within the images captured onboard the satellites of the Earth's surface, and thus must compute a object identification application. An example of an application for such a purpose is IARPA's Function Map of the World, a deep neural network based image detector [36]. The distribution of such deep neural-networks, which has been studied by Teerapittayanon for training purposes [37] with promising results, can provide an efficient breakdown of the detector into component tasks. So, the application in question can be decomposed into a set of tasks with precedence relations, with each task having an output. The constellation features inter-satellite links for the purpose of passing messages containing these outputs, allowing the application to be spread amongst the constellation. We want to find a schedule and assignment for the application on the constellation that minimizes makespan. The realized latency of the system could be significantly decreased if multiple satellites within the constellation could be utilized to process the application. In this case, scheduling towards that end would require accounting for the time-varying nature of the cross-plane inter-satellite links. If one were to use a simple scheduling method, the results might fail to utilize the additional satellites in the constellation to the detriment of the makespan of the application, as we will demonstrate in the example of this case study.

4.1 Modelling parameters

The main focus of this case study is accounting for the time-varying links connecting the satellites. Therefore, we focus our modelling of the problem on representing a compelling example of the behavior of these inter-satellite links. The defining parameters of the case study are divided into three categories. The first category is the defining orbital elements of the constellation. The second category is the application breakdown into composite tasks. The third category is the onboard processor and inter-satellite link characteristics, meant to represent an abstraction of the hardware specifications. Specifically,

the channel bandwidth and effective power to noise ratio for the inter-satellite links. For estimating the completion time of passing a message of a given size over an inter-satellite link, we only consider the theoretical maximum capacity of the channel underlying the link as a function of physical parameters. The resulting time required to pass a message over the links represents a lower-bound on the amount of time required to pass the message. The actual amount of time may also change in real circumstances based on the computer networking paradigm in use, but for a dedicated circuit, the actual time will be proportional. For our case, consider the message size to be proportional to the actual message size in order to compensate for the computer network configuration induced delays. We assume that the proportion is invariant across tasks and machines.

4.1.1 Orbits

In this case study, we base the orbital parameters of the satellite constellation off of the Starlink constellation as described by Handley [34]. Specifically, we consider a constellation consisting of multiple planes of satellites, where the planes are differentiated by longitude of the ascending node, and the satellites within each plane are differentiated by argument of periapsis. The orbits that compose this constellation are circular, and thus the altitude of any satellite within the constellation is constant and equivalent to any other within the constellation. In modelling the orbits for the case study, we use the toolbox Poliastro [38].

4.1.2 Inter-satellite link existence

For the purposes of our case study, we assume the inter-satellite links within this constellation to be line-of-sight (LOS). Thus, in order to determine whether or not two satellites are connected, we consider whether or not the direct path between the two satellites in question intersects the celestial body. In modelling the problem for scheduling, we only consider links that exist for the entirety of the time window of interest. When the orbital

planes are composed of circular orbits, the existence of a given link at a given point in time can be found knowing the distance between satellites and the altitude of the constellation. If the inter-satellite distance is greater than $2\sqrt{r_s^2 - R^2}$, where r_s is the orbital radius of the satellites, and R is the radius of the celestial body, then the inter-satellite direct path intercepts a spherical approximation of the celestial body, and thus the inter-satellite link does not exist. There could be several methods to solve problems in which the existence of inter-satellite links is dynamic, however that is beyond the scope of this work.

4.1.3 Inter-satellite link capacity

The capacity of the inter-satellite links varies as a function of the distance between the satellites. For two satellites connected by and inter-satellite link in two different orbital planes, the distance of the satellites will vary throughout the orbital period, and therefore the capacity of the inter-satellite link will vary throughout the orbital period. Using a simplified path loss model as found in Goldsmith [39], the capacity of a channel is

$$C = B \log_2(1 + \frac{P_t K}{N} \frac{1}{d^\gamma}), \tag{4.1}$$

where d is the path length, B is the bandwidth of the channel, P_t is the transmit power, K is the free-space path loss w.r.t. one unit of d, and γ is the path-loss exponent.

For our purposes, the maximum information that can be transmitted over a link during time interval $[t_0, t_1]$ with a given time-varying channel capacity $C(t)$ as found by using a time-dependent distance $d(t)$ in (4.1) is

$$I(t_0, t_1) = \int_{t_0}^{t_1} C(\tau) d\tau. \tag{4.2}$$

Thus, the earliest possible time to complete the passing a message of size S, starting at time

t_0 can be found by solving

$$\min_{t \geq t_0} \quad t \tag{4.3}$$

$$s.t. \quad S \leq \int_{t_0}^{t} C(\tau) d\tau. \tag{4.4}$$

For this case study, $d_{i,j}(t)$ is the distance between two communicating satellites i and j, given as:

$$d_{i,j}(t) = \|p_i(t) - p_j(t)\|_2, \tag{4.5}$$

where $p_i(t), p_j(t)$ are the positions of satellite i and j in the earth centered inertial frame at time t. In this case study, the hardware underlying the inter-satellite links is uniform across the constellation, so the capacity of the inter-satellite link between satellites i and j at time t becomes

$$C_{i,j}(t) = B \log_2(1 + \frac{P_t K}{N} \frac{1}{d_{i,j}^{\gamma}(t)}) \tag{4.6}$$

using a simplification of $\gamma = 2$ and $K = 1$. Thus, the time it would take to pass the results of a task whose output size is S would be

$$\min_{t \geq t_0} \quad t \tag{4.7}$$

$$s.t. \quad S \leq \int_{t_0}^{t} B \log_2(1 + \frac{P_t}{N} \frac{1}{d_{i,j}^2(\tau)}) d\tau. \tag{4.8}$$

With the above, we can find the time at which a message passing operation of size S would be complete on the inter-satellite link between satellites i and j.

4.2 Conversion to scheduling problem

In order to solve the scheduling problem for the satellite constellation case study, we need to not only schedule the processing of the constituent tasks of the application, but also the message passing of task outputs between assigned machines. In order to accomplish this, we treat the inter-satellite links as machines and the passing of outputs between processing tasks as tasks themselves. Therefore, the scheduling problem has precedence related tasks without unit processing time that must be scheduled and assigned to nonuniform and heterogeneous machines with machine precedence.

Initially, we know the orbital elements of the satellites, a time window of interest, the application to be scheduled with its component tasks, the estimated times to execute those tasks on the satellites, and the task output sizes.

In formulating the scheduling problem, there are five significant steps. First, we need to find the paths of the satellites during the scheduling window. Second, we need to determine which of the inter-satellite links are relevant during the scheduling window. Third, we need to find how the parameters of the relevant inter-satellite links vary during the window. Fourth, we need to define the completion time functions of both the new processing and communication tasks. Finally, we need to construct a new precedence graph incorporating the communications tasks.

We first propagate the satellites throughout the time window in order to determine their positions. Once the positions are known, the properties of the inter-satellite links can be determined. For our case, we only check that the link does not intersect the celestial body that the constellation orbits. Once we know the extant links, we can construct a machine precedence graph. We also construct a modified task precedence graph, including the communication tasks. We must determine the correct machine and task typing tensor to match communication tasks to links and processing tasks to processors. Finally, we must compute the sampled completion time functions for the tasks, using our model of inter-satellite

channel capacity for communication tasks, and constant processing time for the processors. If we cannot find the completion time, we assume it to be outside of the window of interest and set it to be marginally greater than the length of the window.

Since the problem has nonuniform machines, we must use one of the three methods presented in Section 3.4 in order to solve for a schedule using the approximation. For this case study, we use the third method. We do this because the first two methods do not approximate well in the presence of a large number of machines where the best isolated approximation is equivalent to the worst case processing time approximation. The program

used to solve the scheduling problem for this case is

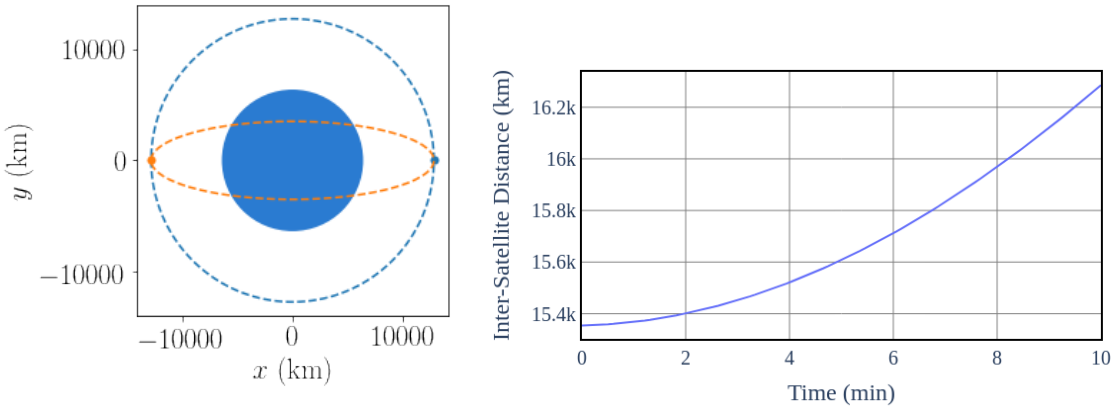
$$
\begin{aligned}
\min_{\boldsymbol{p}} \min_{\boldsymbol{s},\boldsymbol{\sigma},\boldsymbol{x},\boldsymbol{\epsilon},\boldsymbol{\gamma}} \max_{i} \quad & C_i \\
\text{s.t.} \quad s_i &\geq 0 && \forall i \in N \\
s_j - C_i - W(\sigma_{i,j} - 1) &\geq 0 && \forall i, j \in N, i \neq j \\
\sigma_{i,j} + \sigma_{j,i} &\leq 1 && \forall i, j \in N, i \neq j \\
C_i &\leq H && \forall i \in N \\
C_i &= k_1 s_i + k_2 && \forall i \in N \\
\sigma_{i,j} &\geq a_{i,j} && \forall i, j \in N, i \neq j \\
p_i &= \sum_{k=1}^{M} k x_{i,k} && \forall i \in N \\
\epsilon_{i,j} + \epsilon_{j,i} &\leq 1 && \forall i, j \in N, i \neq j \\
\epsilon_{i,j} + \epsilon_{j,i} + \sigma_{i,j} + \sigma_{j,i} &\geq 1 && \forall i, j \in N, i \neq j \\
p_j - p_i - \epsilon_{i,j} M &\leq 0 && \forall i, j \in N, i \neq j \\
p_j - p_i - 1 - (\epsilon_{i,j} - 1) M &\geq 0 && \forall i, j \in N, i \neq j \\
x_{i,k} &\leq u_{i,k} && \forall i \in N, k \in M \\
x_{i,h} - \gamma_{i,j,h,k} &\geq 0 && \forall i, j \in N, i \neq j, h, k \in M \\
x_{j,k} - \gamma_{i,j,h,k} &\geq 0 && \forall i, j \in N, i \neq j, h, k \in M \\
x_{i,h} + x_{j,k} - 1 - \gamma_{i,j,h,k} &\leq 0 && \forall i, j \in N, h, k \in M \\
a_{i,j} \gamma_{i,j,h,k} &\leq b_{h,k} && \forall i, j \in N, h, k \in M,
\end{aligned}
\tag{4.9}
$$

which is a combination of constraints as presented thus far. The total number of constraints of this program is $N(2NM^2 + 2(N-1)M^2 + 8N - 3)$, where N is the number of tasks to be scheduled, and M is the number of machines to be scheduled on. There are $N(N(M^2 + 2) + M + 1)$ variables, of which $N(N(M^2 + 2) + M)$ are integer variables. Using method three, the size of the search set for the outer program is M^N, so without constraints the

program is computed M^N times. However, we can constrain the programs to be computed by translating the constraints on x into the choice of p. The number of tasks in terms of the case study parameters is $N = |V_{\text{app}}| + |E_{\text{app}}|$, where V_{app} is the vertex set of the precedence graph of the application and E_{app} is the edges set of the precedence graph of teh application. The number of machines in terms of the case study parameters is $M = 2S + L$, where S is the number of satellites and L is the number of inter-satellite links. In order to solve this, the quadratic program (2.46) must be solved over the approximation window first, which grows in the number of constraints linearly with the number of time samples.

4.3 Example

Consider two planes of one satellite each, orbiting the Earth at an altitude of 12756 km and inclination of 53 degrees. The time window of interest is 10 minutes, which corresponds to roughly $\frac{1}{44}$ of the orbital period. We sample 1000 equally spaced time steps from the time window.



(a) Orbits

(b) Inter-Satellite Distance

Figure 4.2: Visualization of the paths and relevant physical parameters of the case study example constellation.

We consider scheduling an application composed of three tasks related by a graph as shown in Figure 4.3, and the values of the application parameters are listed in Table 4.1.

The values of the inter-satellite link parameters are listed in Table 4.2. The self-loop

Table 4.1: Case Study: Application Parameters

Parameter	Task 1	Task 2	Task 3
Processing Time (min)	2.239	1.983	2.079
Output Size (KB)	0.333	0	0

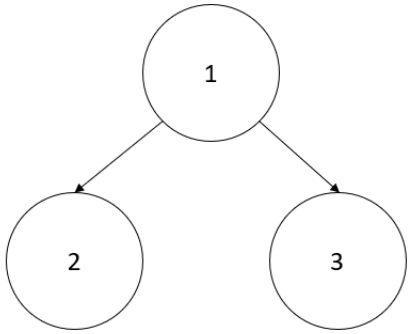


Figure 4.3: Breakdown of application into tasks and precedence relations.

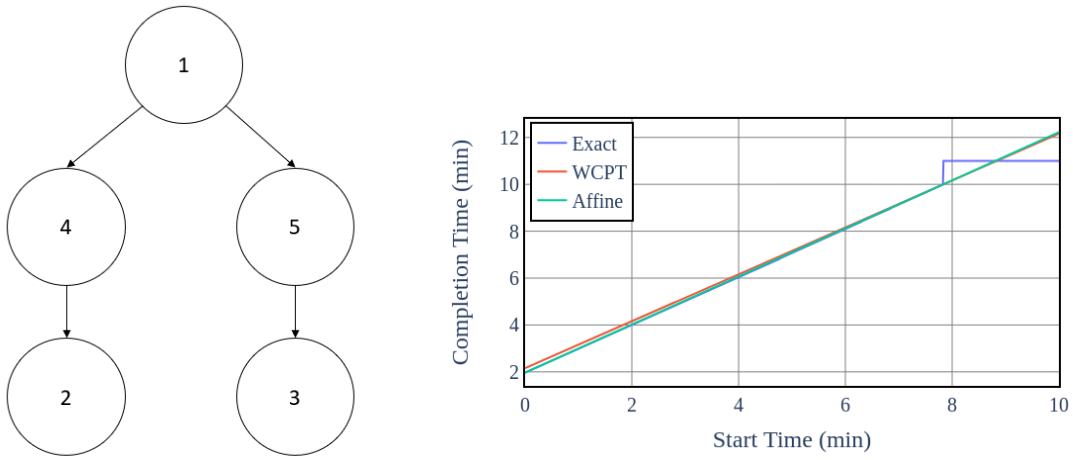
Table 4.2: Case Study: Link Parameters

Parameter	Value
Bandwidth (kHz)	536
Transmit Power to Noise Ratio	10000
Self-Loop Distance (km)	200

distance is the distance in kilometers used to calculate the link capacity between a given satellite and itself for purposes of computing communication task machine parameters.

Once we have converted this to a scheduling problem using the process described in Section 4.2, we are left with 5 tasks and 5 machines. The three additional machines represent the links between processors. Machine 3 is the self-loop for the first satellite onboard processor, represented by Machine 1, and Machine 5 is the same for Machine 2. Machine 4 represents the inter-satellite link between the two satellites, connecting Machine 1 and Machine 2. The two additional tasks represent message passing of the processing task outputs. Task 4 represents the message that must be passed between the machine Task 1 is assigned to and the machine Task 2 is assigned to, while Task 5 represents the same for Task 1 and Task 3. The new precedence graph relating these five tasks are shown in Figure 4.4a. The corresponding approximations and completion times of the two message passing

tasks across the three communications links are shown in Figure 4.4b.



(a) Task Precedence Graph

(b) Completion Time

Figure 4.4: Case study conversion task precedence graph and completion time functions. Note how the completion time has a step where the solution for the exact completion time is no longer within the scheduling window.

The resulting schedules when using the WCPT approximation and the affine over-approximation resulting from Method 3 are listed in Tables 4.3 and 4.4 respectively. The schedules are shown in Figures 4.5a and 4.5b.

Table 4.3: Case Study: WCPT Schedule

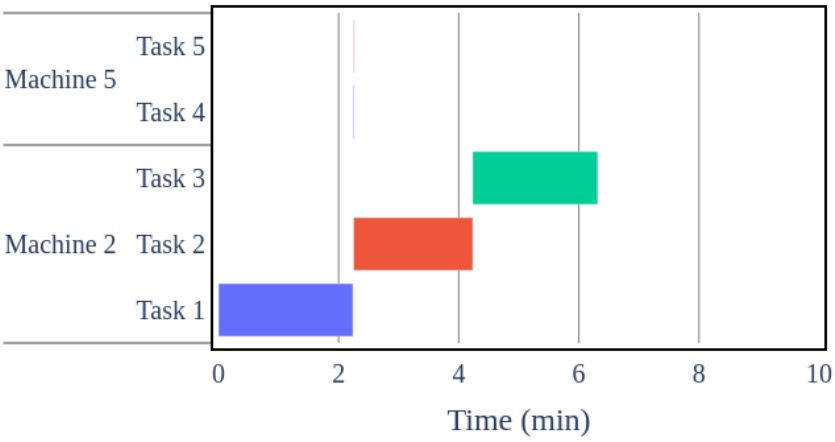
Task	1	2	3	4	5
Start Time (min)	0	2.249	4.231	2.239	2.249
Machine	2	2	2	5	5

Table 4.4: Case Study: Affine Schedule

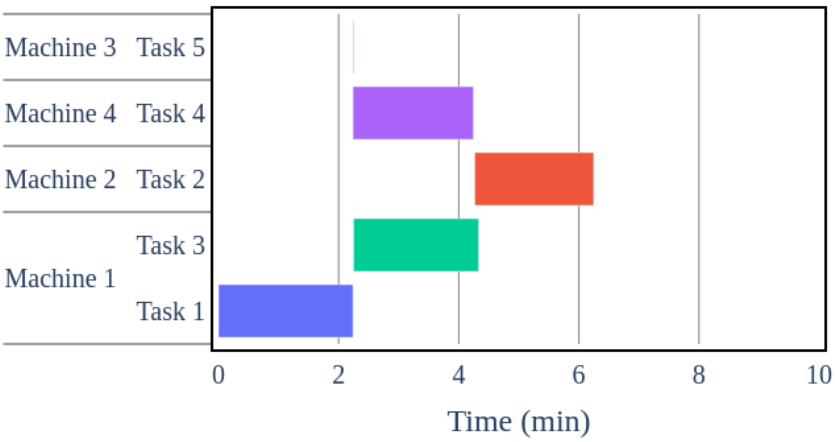
Task	1	2	3	4	5
Start Time (min)	0	2.249	4.231	2.239	2.249
Machine	0	1	0	4	3

The resulting makespans are listed in Table 4.5. As expected, the WCPT makespan is greater the affine over-approximation makespan.

Also, note the difference in which machines are utilized by the two schedules. The schedule generated with the WCPT approximation only uses one of the two satellites, while



(a) WCPT Schedule



(b) Approximation Schedule

Figure 4.5: Visualization of the case study example schedules.

Table 4.5: Case Study: Makespans

Method	Makespan (min)
WCPT	6.310
Affine	6.249

the schedule generated with the affine over-approximation uses both of the satellites. If these schedules were converted to ordered assignments, the makespan of using the affine

over-approximation would be likely be even lesser than using the WCPT approximation when executed on the constellation.

CHAPTER 5
CONCLUSION

In this thesis, we have presented a framework for scheduling in time-varying environments. The framework is composed of a representation for completion time behavior, an approximation method for such behavior, and a mixed-integer program to solve for a schedule using the approximation. By scheduling using our approximation method instead of the WCPT approximation, we have access to a larger number of feasible problems for the base problem class. For certain more complex problems, the resulting schedule using our approximation method can still yield improvement over the WCPT approximation as well as providing a tractable scheme when the exact solution is intractable, as shown by the numerical results in Chapter 3. This can also be seen for practical problems, such as in the satellite constellation case study we presented, where scheduling using the approximation method leads to a materially different outcome.

The results from this thesis can be extended in a number of ways. Approximation and scheduling methods that make use of higher-order convex polynomials in order to better approximate the completion time function can provide significant improvement over the first-order affine approximation presented. The results with relation to minimizing makespan as the optimality criterion should also be applicable to other completion time based objectives, such as minimizing lateness. A method can be found to schedule for a problem with a set of independent, cyclic real-time tasks. For the satellite constellation case study, a closed-form solution for the positions can be found, and thus communication completion time functions. Also, for the case study, situations in which the satellite operations are scheduled may also be of interest, such as dynamic voltage and frequency scaling for on-board processors that vary with the exposure of the satellite to sunlight. Finally, extension of the scheduling method introduced in this thesis to other applications involving groups of

networked vehicles, such as a group of UAVs, holds promise.

REFERENCES

[1] NVIDIA. (2020). Jetson agx xavier.

[2] Qualcomm. (2020). V2x.

[3] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[4] Amazon. (2018). Announcing aws robomaker.

[5] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea, "Rapyuta: The roboearth cloud engine," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 438–444.

[6] F. J. Ros, J. A. Martinez, and P. M. Ruiz, "A survey on modeling and simulation of vehicular networks: Communications, mobility, and tools," *Computer Communications*, vol. 43, pp. 1–15, 2014.

[7] A. Razi, F. Afghah, and J. Chakareski, "Optimal measurement policy for predicting UAV network topology," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*, IEEE, 2017.

[8] N. Bui, M. Cesana, S. A. Hosseini, Q. Liao, I. Malanchini, and J. Widmer, "A survey of anticipatory mobile networking: Context-based classification, prediction methodologies, and optimization techniques," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1790–1821, 2017.

[9] G. Best, M. Forrai, R. R. Mettu, and R. Fitch, "Planning-aware communication for decentralised multi-robot coordination," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018.

[10] S. A. Zanlongo, A. C. Wilson, L. Bobadilla, and T. Sookoor, "Scheduling and path planning for computational ferrying," in *MILCOM 2016-2016 IEEE Military Communications Conference*, IEEE, 2016, pp. 636–641.

[11] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Communication-aware scheduling of serial tasks for dispersed computing," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1330–1343, 2019.

[12] S. Li, Z. Zheng, W. Chen, Z. Zheng, and J. Wang, "Latency-aware task assignment and scheduling in collaborative cloud robotic systems," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, IEEE, 2018.

[13] D. Wang, Z. Liu, X. Wang, and Y. Lan, "Mobility-aware task offloading and migration schemes in fog computing networks," *IEEE Access*, vol. 7, pp. 43 356–43 368, 2019.

[14] A. Rahman, J. Jin, A. Cricenti, A. Rahman, and M. Panda, "Motion and connectivity aware offloading in cloud robotics via genetic algorithm," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, IEEE, 2017.

[15] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," in *Annals of discrete mathematics*, vol. 5, Elsevier, 1979, pp. 287–326.

[16] J. D. Ullman, "Np-complete scheduling problems," *Journal of Computer and System sciences*, vol. 10, no. 3, pp. 384–393, 1975.

[17] C.-C. Shen and W.-H. Tsai, "A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion," *IEEE Transactions on Computers*, vol. 100, no. 3, pp. 197–203, 1985.

[18] J. B. Sinclair, "Efficient computation of optimal assignments for distributed tasks," *Journal of Parallel and Distributed Computing*, vol. 4, no. 4, pp. 342–362, 1987.

[19] H. El-Rewini and T. G. Lewis, "Scheduling parallel program tasks onto arbitrary target machines," *Journal of parallel and Distributed Computing*, vol. 9, no. 2, pp. 138–153, 1990.

[20] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys (CSUR)*, vol. 31, no. 4, pp. 406–471, 1999.

[21] T. Davidovic, L. Liberti, N. Maculan, and N. Mladenovic, "Towards the optimal solution of the multiprocessor scheduling problem with communication delays," 2007.

[22] S. Venugopalan and O. Sinnen, "Ilp formulations for optimal task scheduling with communication delays on parallel systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 1, pp. 142–151, 2015.

[23] A. Metzner, M. Franzle, C. Herde, and I. Stierand, "Scheduling distributed real-time systems by satisfiability checking," in *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA05)*, IEEE.

[24] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM computing surveys (CSUR)*, vol. 43, no. 4, p. 35, 2011.

[25] Dar-Tzen Peng, K. G. Shin, and T. F. Abdelzaher, "Assignment and scheduling communicating periodic tasks in distributed real-time systems," *IEEE Transactions on Software Engineering*, vol. 23, no. 12, pp. 745–758, 1997.

[26] T. Abdelzaher and K. Shin, "Combined task and message scheduling in distributed real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 11, pp. 1179–1191, 1999.

[27] S. S. Craciunas and R. S. Oliver, "Combined task- and network-level scheduling for distributed time-triggered systems," *Real-Time Systems*, vol. 52, no. 2, pp. 161–200, 2015.

[28] S. Gertphol and V. Prasanna, "MIP formulation for robust resource allocation in dynamic real-time systems," in *Proceedings International Parallel and Distributed Processing Symposium*, IEEE Comput. Soc.

[29] I. Hamaz, L. Houssin, and S. Cafieri, "The Cyclic Job Shop Problem with uncertain processing times," in *16th International Conference on Project Management and Scheduling (PMS 2018)*, Rome, Italy, Apr. 2018.

[30] M. Neely, E. Modiano, and C. Rohrs, "Dynamic power allocation and routing for time varying wireless networks," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, IEEE.

[31] E. Balas, "Disjunctive programming and a hierarchy of relaxations for discrete optimization problems," *SIAM Journal on Algebraic Discrete Methods*, vol. 6, no. 3, pp. 466–486, 1985.

[32] H. Santos and T. Toffolo, *Python mip: Version 1.8.1*, Apr. 2020.

[33] J. Forrest, T. Ralphs, S. Vigerske, LouHafer, B. Kristjansson, jpfasano, EdwinStraver, M. Lubin, H. G. Santos, rlougee, and M. Saltzman, *Coin-or/cbc: Version 2.9.9*, version releases/2.9.9, Jul. 2018.

[34] M. Handley, "Delay is not an option: Low latency routing in space," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, 2018, pp. 85–91.

[35] W. Commons. (2019). 1,584 satellites will be into 72 orbital planes of 22 satellites each, inclination 53. File:Starlink SpaceX 1584 satellites 72 Planes 22each.png.

[36] M. Pritt and G. Chern, "Satellite image classification with deep learning," in *2017 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, IEEE, 2017, pp. 1–7.

[37] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2017, pp. 328–339.

[38] J. L. C. Rodrguez, A. Hidalgo, S. Bapat, Jorge, N. Astrakhantsev, C. Eleftheria, D. Lubin, A. L. Mrquez, E. Selwood, Meu, P. R. Robles, H. Eichhorn, A. Rode, H. Garg, aOrionis, H. Goyal, AntoniiaK, Angala, A. Yusril, O. Streicher, I. C. Fernndez, F. P. Schmidt, D. Raina, R. Malhotra, M. Rossman, M. Pilosov, J. E. MEng, V. Naik, Sky, and M. Kaufmann, *poliastro/poliastro: poliastro 0.13.1 (Christmas '19 edition)*, version v0.13.1, Dec. 2019.

[39] A. Goldsmith, *Wireless communications*. Cambridge university press, 2005.